



**HAL**  
open science

## Integration of SoT packages into ROS

Thomas Peyrucain

► **To cite this version:**

| Thomas Peyrucain. Integration of SoT packages into ROS. Robotics [cs.RO]. 2020. hal-03144164

**HAL Id: hal-03144164**

**<https://laas.hal.science/hal-03144164v1>**

Submitted on 17 Feb 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

**AIP**  **PRIMECA**

&



# Integration of SoT packages into ROS

*A dissertation submitted in partial fulfilment of the requirements  
for the degree of Engineer from the ESIGELEC School*

## **ABSTRACT**

As a part of the RIO project that wants to broadcast the usage of ROS into the industry, I am working at releasing 10 years of developed software for humanoid robot and the TIAGO robot into ROS. This work has been started and aborted in the past because the technology and the market expectation were not ready. My internship will then help the Robot community to use those advancement made by the laboratory especially RIO training that want to create a ROS training platform. I will also have to create a tutorial for future maintenance and new packages releases to help the laboratory staff gain time and maintain my work even with the standard evolution.

Keywords:

TIAGO; Robotics; humanoid robots; LAAS – CNRS; rob4fam

## ACKNOWLEDGEMENTS

I want to thanks all the staff from AIP-Primeca and LAAS-CNRS even though we did not meet in real life due to covid situation.

A special thanks to Olivier STASSE who helped me understand the packages and Guilhem Saurel who checked and validated all my pull request on GitHub and made the releases of the packages so I could work properly. Thank you for the time you gave me.

Thank you, Mr Tang, who supported me during this internship and gave me the lines to follow for the thesis.

Also a special thanks to ROS developers who checked and validated my code to put it on the ROS build farm.

I want to thanks my family that encouraged me during this uncertain time.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287. The

[RIO project](#) started in Jan 2020 and will end in Dec 2020.



# TABLE OF CONTENTS

1	Introduction.....	1
1.1	State of the art.....	1
1.2	Description of the LAAS-CNRS.....	7
1.2.1	Gepetto team.....	8
1.3	Description of the project: RIO (ROS in Occitanie).....	9
1.3.1	ROSin.....	9
1.3.2	Context.....	9
1.3.3	Problem statement and ambitions.....	9
1.3.4	Work plan.....	10
1.3.5	My project.....	12
1.4	Description of the Joint Robotics Laboratory (JRL) between CNRS (France) and AIST (Japan).....	13
1.5	Description of the joint laboratory between AIRBUS and LAAS-CNRS: Rob4Fam.....	14
2	Stack of Tasks (SoT).....	15
2.1	Dynamic-graph.....	15
2.2	Dynamic-graph-python.....	16
2.3	Dynamic-graph-tutorial.....	16
2.4	Sot-core.....	16
2.5	Sot-tools.....	16
2.6	Sot-application.....	16
2.7	Task-Space Inverse Dynamics (TSID).....	16
2.8	Sot-dynamic-pinocchio.....	17
2.9	Sot-pattern-generator.....	17
2.10	Sot-torque-control.....	17
2.11	Sot-talos.....	17
2.12	Sot-tiago.....	17
3	ROS ecosystem.....	18
3.1	ROS build farm.....	18
3.1.1	Jenkins.....	18
3.1.2	Rosdistro / ROS versions.....	19
3.1.3	CMake.....	19
3.1.4	Catkin.....	19
4	Methodology.....	21
4.1	Understand the package dependencies.....	21
4.2	Fix warnings.....	22
4.3	Create the files.....	27
4.3.1	Package.xml.....	27
4.3.2	Tracks.yaml.....	29
4.4	Prerelease with Docker.....	31
4.5	Release.....	32

5 Tutorial.....	33
5.1 Release for the first time.....	33
5.1.1 Github release repository.....	33
5.1.2 Authorizing Bloom to generate pull request on the rosdistro repository.....	35
5.1.3 Release.....	37
5.1.4 Pull request.....	39
5.1.5 Validation of your pull request by a ROS deputy.....	41
5.2 Maintenance of packages.....	42
5.3 Technical discussion.....	42
6 Issues.....	43
6.1 First release of dynamic-graph.....	43
6.1.1 Warnings.....	43
6.1.2 Missing dependencies.....	44
6.1.3 Submodule issues.....	44
6.1.4 Different standard between LAAS-CNRS and ROS.....	46
6.1.5 Time delay.....	47
6.2 Technical discussion.....	48
7 Conclusion and future work.....	51
Figure 1-1 HRP-2 robot.....	2
Figure 1-2 Pyrène.....	3
Figure 1-3 Power law-based closed-loop control.....	6
Figure 1-4 Description of the LAAS-CNRS research teams.....	7
Figure 1-5 GEPETTO team.....	8
Figure 1-6 Training centre for the RIO project.....	11
Figure 1-7 List of the SoT packages.....	12
Figure 1-8 Experiment on the HRP-2 robot.....	13
Figure 1-9 Collaborators from left to right: Ali Charara (CNRS-INS2I), Liviu Nicu (LAAS-CNRS), Christophe Giraud (CNRS Occitanie Ouest), Sébastien Boria (Airbus, co-director of Rob4Fam), Olivier Stasse (CNRS, scientific responsible Rob4Fam), Patrick Vigié (Airbus).....	14
Figure 1-10 Interface of the website.....	31
Figure 1-11 ROS distro available on the website for prerelease.....	32

Figure 1-12 Creation of a GitHub repository for the release of TSID.....33

Figure 1-13 Jenkins site that shows the unstable release due to warnings.....44

Figure 1-14 Pull request to fix the issue.....45

## **LIST OF TABLES**

Table 1-1 Table of work packages.....10

Table 1-2 Table of project timetable.....10

## **LIST OF EQUATIONS**

## **LIST OF ABBREVIATIONS**

ROS	Robot Operating System
SoT	Stack of tasks
Rob4Fam	Robots For the Future of Aircraft Manufacturing
JRL	Joint Robotics Laboratory
CNRS	Centre national de la recherche scientifique (Translation from french: National centre of the scientific research)
AIST	National Institute of Advanced Industrial Science and Technology
TSID	Task-Space Inverse Dynamic

# 1 Introduction

## 1.1 State of the art

In robotic, the software applications are very heterogenous and imply expertise in various areas: control, mechatronics, computer science, computer vision, signal processing and artificial intelligence. It results in code that is hard to use due to its size and because of the complexity of the link between packages. The software is written in C++ due to performance needs. Through the years tools were developed to help robot software developers to do continuous integration more easily. Tools to build packages were created to maintain the dependencies. Such a system is called a superbuild.

Many laboratories created their ecosystem to develop their packages and be able to use the code developed by the researchers. For instance, in LAAS Robotpkg was created: <http://robotpkg.openrobots.org/>

Robotpkg was created to centralize the maintenance and development of code inside the laboratory. It creates binaries that can be easily installed by external collaborators working with the LAAS. This is realized by configuring, compiling and installing the software on a set of Operating Systems. The currently targeted are Unix variants: ubuntu, CentOS, Debian, NetBSD. Unfortunately, robotpkg is not the most well-known package system in the robotics community. In the robotics community, the main tool used is ROS.[1]

More precisely the packages I need to implement into the ROS build farm are from the Stack of tasks (SoT). It is a GitHub organization currently managed by the Gepetto team at LAAS. Those packages are mainly developed for efficient motion generation applied to humanoid Robots.

The main structure of the Stack of tasks is described in this article [2]. It is a versatile implementation of general inverse kinematics that allow dynamic modification of the graph of computation, reference generation and task switching. This structure is allowing to reuse software blocks and create control framework incrementally. The challenge was to develop a framework without developing a completely new language and provide efficient integration in middleware like CORBA. The functionalities provided by the SoT are :



- Factory of entities to load classes, create/destroy entities, run scripts, triggers computation.
- Entity that takes a string in input and output methods for dynamical models or even walking.
- Signals that communicate between entities
- Features that receive the values of the robot and compute the Jacobian
- Tasks that use features to provide a control law
- Stack of tasks that compute those control laws



**Figure 1-1 HRP-2 robot**

Thanks to those features researchers of the repository succeeded to implement this method on the robot HRP-2 to do force-control based interaction with a human, force-control based interaction with a human while walking and climbing stairs.

In [3] a Real-time Nonlinear Model Predictive Control (NMPC) is proposed for the humanoid robot HRP-2. It generates a walking pattern using the position and orientation of the feet.

This article improved existing methods that were time-consuming and no good for real-time simulation thanks to observations and well-tuned parameters :

- Only one iteration of the algorithm was needed to obtain a reasonable solution.
- Nonlinear constraints that can take into account obstacles
- A dynamical filter that takes the whole body into account
- The method proposed is based on the linear pendulum with simplified hypotheses (Angular momentum of all the joints equal to 0; The robot Center of Mass evolves horizontally; the normal of the contact forces have to be collinear).

This method reduced the time of the walking pattern generator to 2ms.

[3] described a humanoid walking pattern generator software that helps to port easily walking algorithm on the HRP-2 robot. Different walking algorithms are implemented like toe-joint, stepping over obstacles as large as 15cm, quadratic programs thanks to the software modularity.

Thanks to the Geppetto team expertise in humanoid robots gained through the years of experimentation with HRP-2 robot a robot was specified and built by PAL-Robotics, called Pyrène. The first prototype of the TALOS series is described in [4]. This robot is meant to push forward on the research because it offers more possibilities than the HRP-2 robot. The robot was specified to have better communication buses, motor size, reduction, kinematics, range of motion and highly redundant sensors of the actuators. Thanks to that, complex tasks could be performed in the industry. The robot can lift 2 weight of 6 kilos with the arm stretched during 10 minutes. Some advanced were performed to climb stairs. This experiment highlighted some compliance with the robot. In the article, it is suggested that this could be fixed with work on the torque control strategy.



**Figure 1-2 Pyrène**

ROS was another structure created to have broader coverage. In [5], the authors described their philosophy of modular tool-based software development. An efficient package system is proposed where the user specifies dependencies and then the software automatically builds the packages with the right dependencies.

Another approach to packages was created by GitHub where the packages are all checked for dependencies within GitHub. However, this approach does not apply to packages from LAAS because it is limited to JavaScript.

A lot of solutions were created at the same time to tackle the issue of poor compatibilities between robots and “user lock-in” issues. The OROCOS project [6] aims at creating an open software for robot control in real-time. The aim is to cover all robots and be accepted in academia and in the industry to complement big robot companies and help little companies to have robots that perform more complex tasks. The system is designed to have robots from different companies without changing all their process.

Since 2015 patents were created to tackle those issues especially from the USA : [7]–[10]

All this package management raises also security issues. [11] described many attacks that are possible and how to counter them. For example, if a package with a virus inside is released and then downloaded by a user. The virus will be installed on his machine. To avoid that only trusted software need to be downloaded. The software maintainers also need to list all those trusted packages.

To have a better understanding of the packages; I read papers that were using the stack of tasks to perform various tasks. Those papers also helped the improvement of the packages of the stack of tasks.

This article [12] focuses on a landmark-based approach with a constraint to have at least one landmark always visible. It is using high-level motion planning and a stack of visual servoing tasks to generate obstacle-free motion. This work is using the Stack of Tasks packages to do motion generation with a hierarchical inverse kinematics solver. More precisely it is using dynamic-graph, dynamic-graph-python, sot-core, jrl-walkgen, sot-pattern-generator, dynamic-graph-bridge. The developers ported the code on the HRP-2 robot at the CNRS-AIST Joint French-Japanese Robotics Laboratory in Tsukuba. It was first tested with a simulation on a 3D environment on a computer. Then the code was ported to the real robot to test its accuracy. It highlighted a major issue on the sensitivity of the visual processes. If there is a loss of visibility the robot needs to stop before searching again for the landmark. As a new approach compares to other papers on the same field; the constraint of the landmark was directly incorporated at the planning level. The robot model is taken from new research in neuroscience.

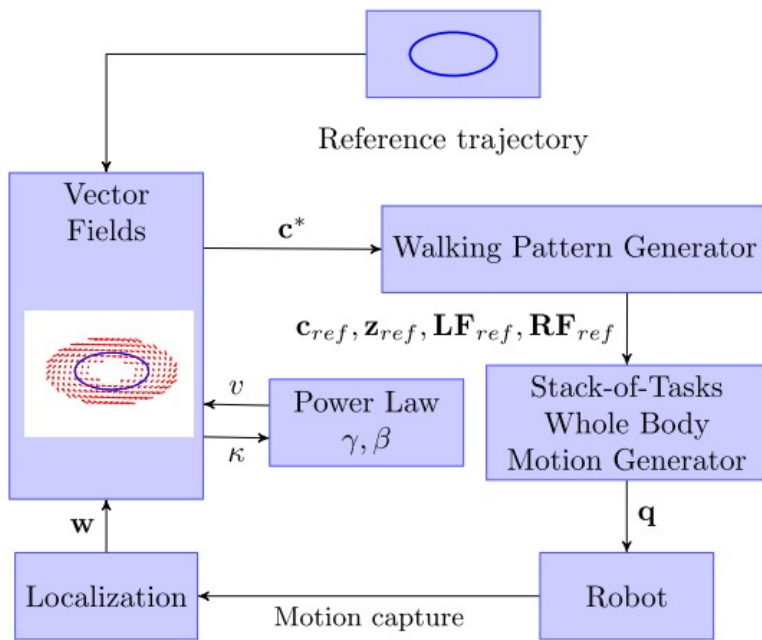
Agimus [13] is a manipulation planning project. The challenge is to take into account the configuration of the object when grasped because once the object is grasped by one or several grippers its position depending on the robot configuration. During the grasping part, the robot needs to not self-collide and maintain its equilibrium. The planning and execution of the manipulation task are composed of three main steps:

- Estimating the configuration of the environment (position of the table and the object)
- Planning a manipulation path from an initial position to a goal position
- Mapping a manipulation path into several sub-tasks

For the estimation of the environment, AprilTags are used on the table and the object with the embedded camera. For the planning manipulation part, a random method is used to find an optimal path. For the mapping part, the researchers are relying on the SoT to generate an instantaneous whole-body control. This method was applied to the Pyrène robot to flip a box upside down while maintaining its equilibrium. At the time of this article, Agimus cannot track failures and cannot recover from it if the object falls to the ground because it needs too much computational power. When applying an open loop to control the robot there was an error of 2 cm between the object and the gripper. It is due to the poor calibration of the robot and the number of joints in the actuation chain. To fix this issue a new AprilTag was added on the robot end-effector.

This article [14] shows the implementation of a torque control strategy on the HRP-2 robot. Even if the robot is built to be position control only, this article describes a torque control architecture to generate its motion while taking into account contact forces. Researchers combined torque control with inverse kinematics to have better motion tracking than position control. This improved accuracy for the same gain. Future work is to review the model and rework it to obtain a better fit. The SoT is used on this project; more precisely dynamic-graph, dynamic-graph-python, sot-core, sot-dynamic-pinocchio, dynamic-graph-bridge.

Fukushima incident showed a lack of applied cases of robots in a non-friendly environment. On this paper [15], the task of the robot was to pull a fire hose. It needed to pull it without self colliding. The experiments applied on an HRP-2 robot put in evidence that the fire hose was creating a drift to the yaw angle of the robot. This drift depends on the weigh of the fire hose. To prevent failure, future work is planned to improve the modelisation of the force of the fire hose by putting it as an external force. Doing that the robot can self-balance thanks to the feedback of this force. This paper uses the SoT, more precisely dynamic-graph, dynamic-graph-python, jrl-walking and dynamic-graph-bridge.



**Figure 1-3 Power law-based closed-loop control**

This article [16] proposes to implement power laws inspired by humans on a humanoid robot. The HRP-2 robot uses this power law to walk along an elliptic trajectory. To control the robot it is using a closed-loop to stay on the trajectory. The location of the robot determine through motion capture is used for the closed-loop. It is controlled like a mobile platform with the x and y velocities and the angular velocity applied to the centre of mass. Constraints

were applied to guarantee the equilibrium of the robot. The centre of mass is constrained around the centre of the support foot. The footsteps are predefined as polyhedra for their motion. To obtain the speed a trajectory was determined by tacking the centre between two opposite local extremum. The power-law  $\beta$  was estimated thanks to nonlinear regression. The result of the experiments showed that using a higher  $\beta$ , the speed could be accelerated. The usage of a humanlike law reduced the drift on the robot around the trajectory on the simulations. This paper uses the SoT, more precisely dynamic-graph, dynamic-graph-python, tsid, sot-core, jrl-walking, sot-pattern-generator and dynamic-graph-bridge.

## 1.2 Description of the LAAS-CNRS

It is a laboratory in Toulouse that is organised in 4 research axes:

- Informatics
- Robotics
- Automatics
- Nanosystems

In this laboratory, it exists 8 main departments and 26 research teams.

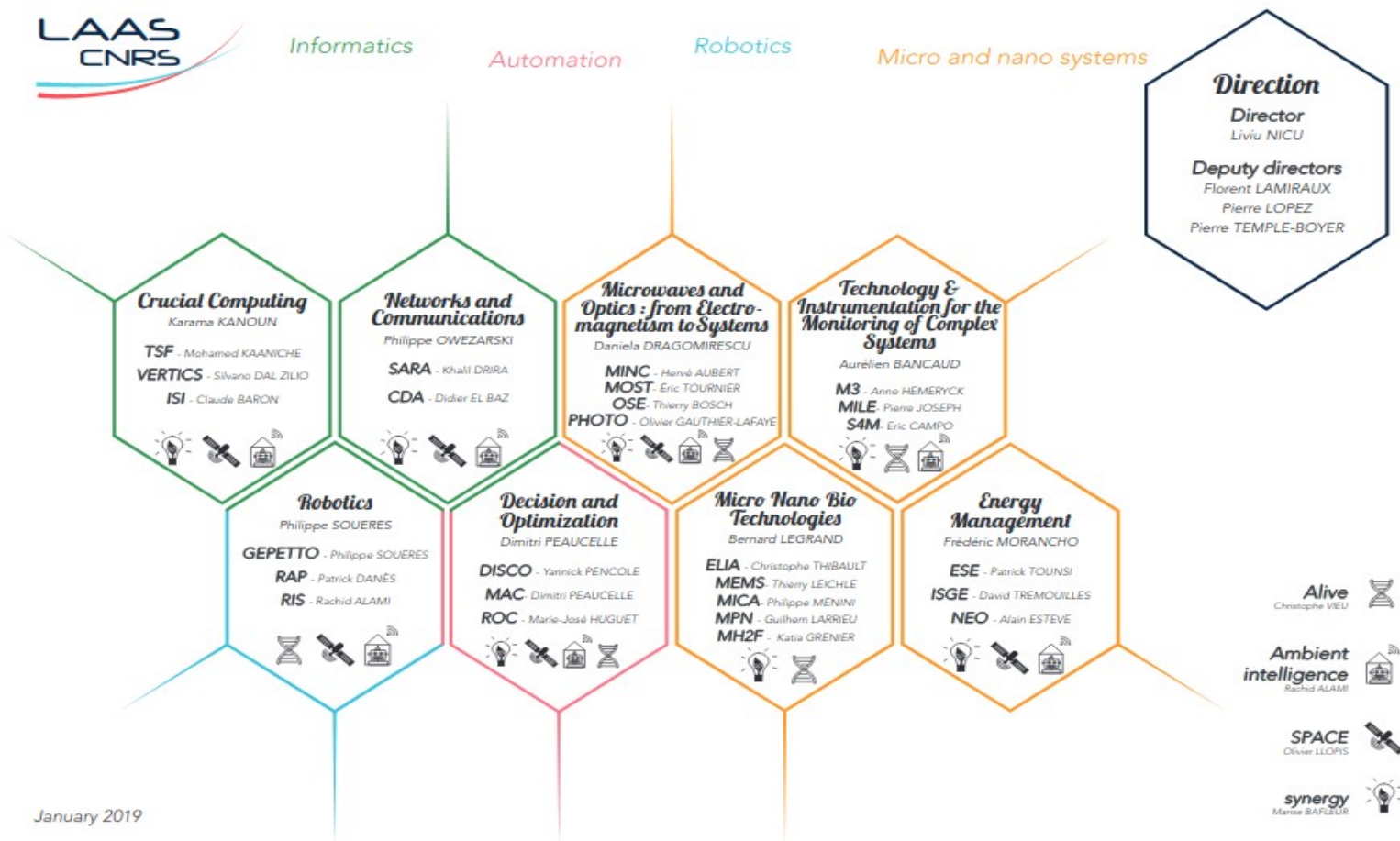


Figure 1-4 Description of the LAAS-CNRS research teams

Inside this laboratory, I am working with the Gepetto team.

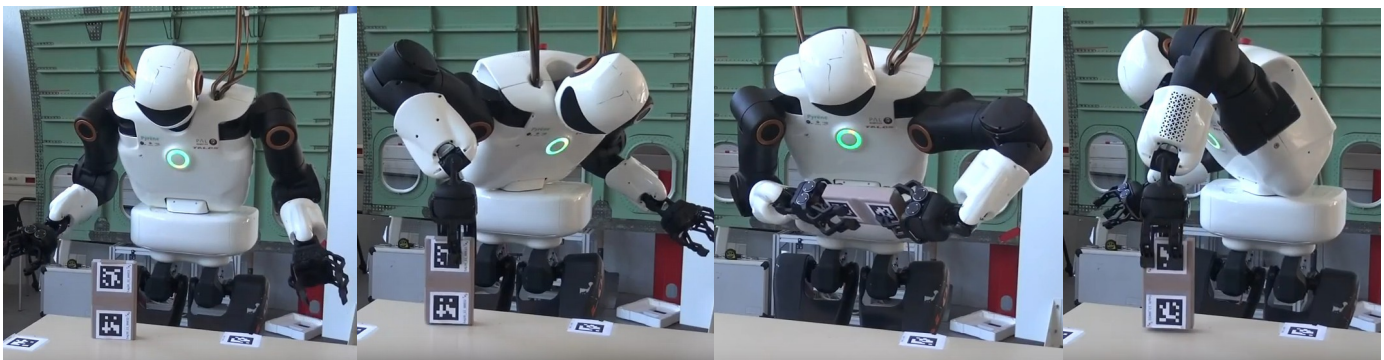
### 1.2.1 Gepetto team



Figure 1-5 GEPETTO team

The Gepetto team is in charge of developing software for humanoid robots and mobile robots. An outcome is the development of scientific methods to understand and generate motion for anthropomorphic structure. Their researches lead to the creation of robots, successful motion planning and achievement of complex tasks. They are working in close collaboration with other companies to develop new solutions that match the needs of the industry.

For example, they created **Agimus** that is a manipulation motion planner involving multiple contacts. It automatically generates the control sequence to execute the manipulation task while keeping the robot balanced. This planner implements this in real-time thanks to the development made on the software Stack-of-Tasks.



## 1.3 Description of the project: RIO (ROS in Occitanie)



### 1.3.1 ROSin

Rio project is funded in the frame of ROSIN a cascade funding project. ROSin is aiming at expanding the usage of ROS in the industry thanks to open-source software. It is a European project that unlocked more than 3 million euros to third parties for ROS-industrial development.

### 1.3.2 Context

Situated in Toulouse the capital of aeronautics and space industry, big and medium companies in aeronautics wants to reduce production cost and to remain competitive by putting effort to robotize and digitalize their production processes. There are 720 companies which cover around 70000 jobs around Toulouse (before COVID 19). In this transition toward Industry 4.0, they lack skilled workers in robotics. This is slowing down the process of robotization. There is also a big university with more than 90000 students and engineering schools. This ecosystem is favourable to a good dynamic of progress in aeronautics.



### 1.3.3 Problem statement and ambitions

The RIO project aims to create a training centre for robotics in Toulouse where they are aiming to train 200-300 students per year to the ROS environment. They want ROS to be more used in the industrial field around Toulouse and beyond.





### 1.3.4 Work plan

This project is a 1-year project that is separated into three main work-packages as described below :

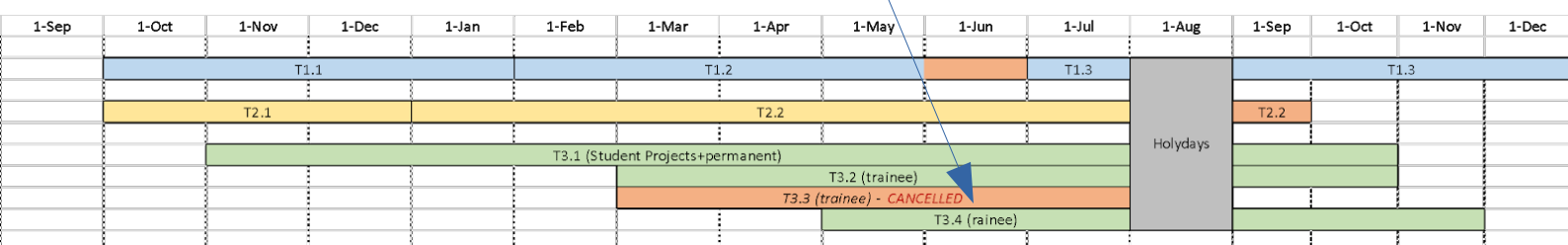
**Table 1-1 Table of work packages**

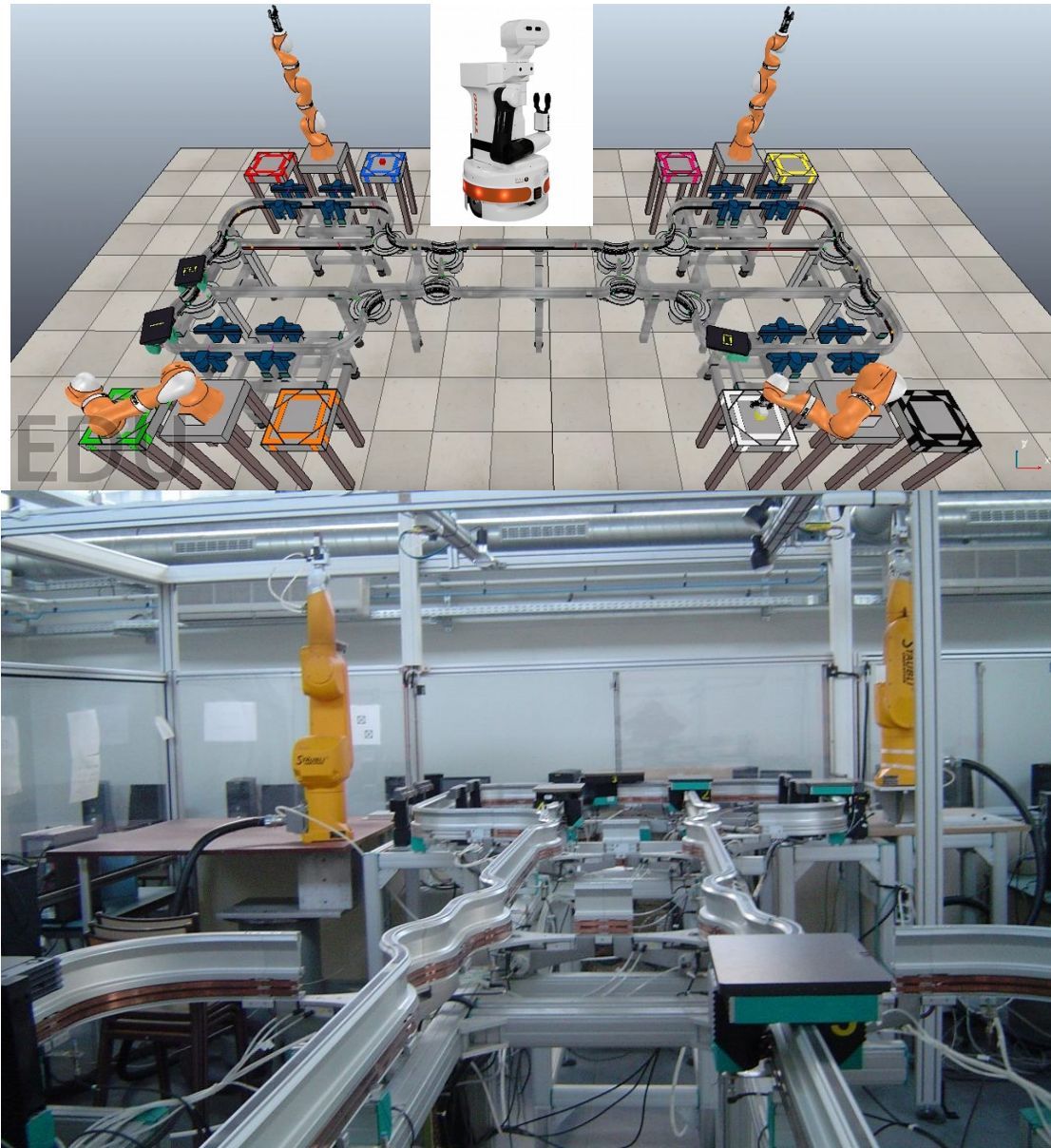
Work-package (WP coordinators)	Tasks	Deliverables
WP1 – Designing new education offers (Briand-Taïx)	Task 1.1: Requirement analysis for the education programmes	A document that analyses the professional skills to target
	Task 1.2: Design of new training offers	The description of a list of new training offers to be validated by our partners’ network
	Task 1.3: Promotion of the RTC and the new education programmes	A web page valorizing the training center and the training offer A mail promotion campaign
WP2 – Training of new instructors (Germa-Stasse)	Task 2.1: Definition of an instructors’ training plan	Definition of a set of training modules for instructors
	Task 2.2: Implementation of the instructors’ training plan and skills certification	A set of skill validation tests
WP3 – Integration of the industrial resources of the AIP platform under ROS (Briand-Germa-Stasse-Taïx)	Task 3.1: Integration of the assembly cell (Montrac, Schneider automatons) (Briand)	POI UC1 implementation
	Task 3.2: Integration of the robotics arms (Stäubli, Kuka) (Taïx)	POI UC2 implementation
	Task 3.3: Integration of the vision systems (Cognex, PTZ Canon) (Germa)	POI UC3 implementation
	Task 3.4: Integration of a mobile manipulator robot (Stasse)	POI UC4 implementation

More precisely my contribution is Task 3.4 which is to integrate the resources of the AIP into the ROS build farm. It consists of packages created by the LAAS research lab to do motion planning for complex robots.

It lasts from the 18 May to the 27 November (24 weeks of work in total).

**Table 1-2 Table of project timetable**





**Figure 1-6 Training centre for the RIO project**

As a training project, AIP Primeca staff wants to create a test base where they can show the power of ROS for industrial applications. They created an assembly line with 4 different robot arms from 4 different constructors and they want to use only ROS to control them all. My contribution will help to integrate one more robot the TIAGO robot inside this test environment as a mobile manipulator. To implement this mobile robot they need the software developed during 10 years the Gepetto team to be put on ROS.

### 1.3.5 My project

For the deliverables, I need to put all the packages on the ROS build farm and create a tutorial to maintain those packages.

I was asked to release those packages for 4 versions/distro of ROS:

- For ROS1: **melodic, noetic**
- For ROS2: **foxy, eloquent**

All the packages are listed below on GitHub (<http://stack-of-tasks.github.io/development.html>) :

## Development dashboard

### Dashboard

Package	Build	Coverage	Issues	Pull Requests
<a href="#">Dynamic Graph</a>	pipeline passed	coverage 93.00%	n/a	n/a
<a href="#">Dynamic Graph Python</a>	pipeline passed	coverage 40.00%	n/a	n/a
<a href="#">Dynamic Graph Tutorial</a>	pipeline passed	coverage 15.00%	n/a	n/a
<a href="#">SoT Core</a>	pipeline passed	coverage 21.00%	n/a	n/a
<a href="#">SoT Tools</a>	pipeline passed	coverage 0.00%	n/a	n/a
<a href="#">SoT Application</a>	pipeline passed	coverage 0.00%	n/a	n/a
<a href="#">TSID</a>	pipeline failed	coverage 40.00%	n/a	n/a
<a href="#">SoT Dynamic Pinocchio</a>	pipeline passed	coverage 4.00%	n/a	n/a
<a href="#">SoT Pattern Generator</a>	pipeline passed	coverage 1.00%	n/a	n/a
<a href="#">Sot Torque control</a>	pipeline passed	coverage 9.00%	n/a	n/a
<a href="#">SoT Talos</a>	pipeline passed	coverage 0.00%	n/a	n/a
<a href="#">SoT Tiago</a>	pipeline passed	coverage unknown	n/a	n/a

**Figure 1-7 List of the SoT packages**

Putting packages on the ROS build farm means that if you have ROS installed on your computer you will have to use:

**sudo apt install ros-<ROS\_version>-<Package\_name>**

And it will build all the needed dependencies with ROS automatically. Then those packages can be directly used with ROS.

## 1.4 Description of the *Joint Robotics Laboratory (JRL)* between CNRS (France) and AIST (Japan)

Created in 2006, JRL is an international laboratory between France and Japan situated in Toulouse. It aims to improve the autonomy of humanoid robots. In the beginning, the laboratory in France and Japan started to develop software for the HRP-2 robot at the same time with 2 identical robots one in France and one in Japan. On one hand, The French team was in charge of the perception, decision, and action, on the other hand, the Japanese team was in charge of mechatronics, teleoperation, and command.



**Figure 1-8 Experiment on the HRP-2 robot**

JRL was broadcasted all around France to create innovative projects with other French laboratories with a broader range of fields like cognitive science and artificial intelligence.

Participants had access to the hardware, software of the robot and to the training to learn how to use it. Thank to that they succeeded to test their program on the real robot to validate their research.

## 1.5 Description of the joint laboratory between AIRBUS and LAAS-CNRS: Rob4Fam

Rob4Fam (*Robots For the Future of Aircraft Manufacturing*) is a joint laboratory of LAAS-CNRS and Airbus for the usage of robotics in aeronautics processes inaugurated the 21 of May 2019.



Figure 1-9 Collaborators from left to right: Ali Charara (CNRS-INS2I), Liviu Nicu (LAAS-CNRS), Christophe Giraud (CNRS Occitanie Ouest), Sébastien Boria (Airbus, co-director of Rob4Fam), Olivier Stasse (CNRS, scientific responsible Rob4Fam), Patrick Vigié (Airbus)



It aims to develop innovative new technologies for production in aeronautics. It means to create adaptative robots that can work with humans and react in real-time with the industrial environment. This joint effort will boost the usage of robots in the industry.

I assisted the weekly and daily meetings of Rob4Fam to better understand the needs of robotics for the aeronautic field and to see the usage of the packages I will be implementing into ROS.

## 2 Stack of Tasks (SoT)

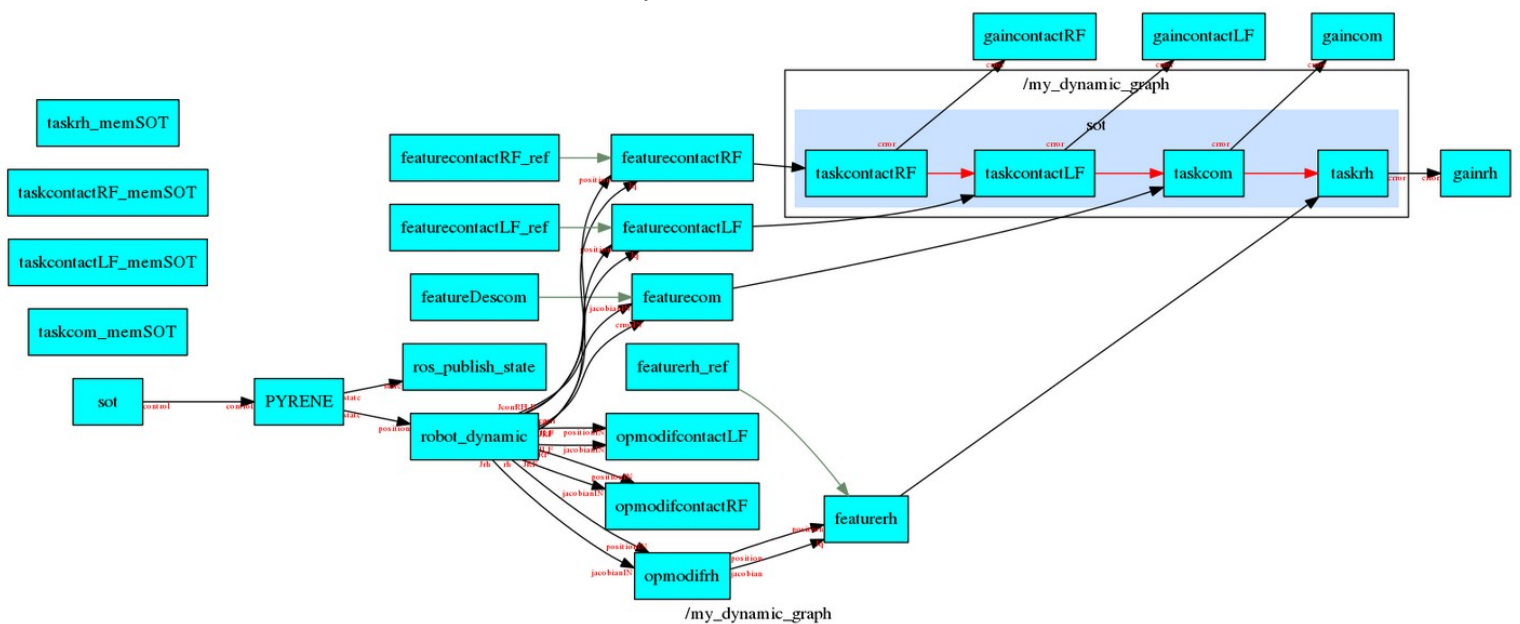
As described in the main paper of the SoT [2]; it is composed of packages that work together as different tasks.

### 2.1 Dynamic-graph

This dynamic-graph is composed of:

- entities (graph nodes)
- signals (input/output of a graph action)
- commands (expand the capabilities of entities)
- factory (manage the nodes)
- pool (handle the instance of a node)

This is a software that is optimized to create a C++ data-flow. It is connecting those “entities” as Simulink does. Thanks to that, it is quite easy to create a full graph for an experiment. This package is the basis of the stack of tasks operation. In a project, it is possible to see this graph with the dot module that we will see is needed on the dependencies.



Here is an example of the dynamic-graph that was used on a project on the Pyrène robot. This structure permits the researchers to do real-time control on humanoid robots.

## 2.2 Dynamic-graph-python

It is the python binding of dynamic-graph that permit to use python to create the dynamic-graph control. It is recommended to use Python rather than C++ to create the graph.

## 2.3 Dynamic-graph-tutorial

It is a step by step tutorial to learn how to create entities on dynamic-graph. It is a simple application of the dynamic-graph package. Dynamic-graph and dynamic-graph-python need to be installed to follow this tutorial.

## 2.4 Sot-core

Sot-core is a software that is using dynamic-graph to define and solve hierarchical tasks.

## 2.5 Sot-tools

It is adding entities and scripts to the SoT framework:

- Cubic interpolation
- OpenHRP joint trajectory file format (Seqplay)
- Quaternion and SE(3) computation implemented in Python

## 2.6 Sot-application

This package provides python initializations scripts for the Stack of Tasks.

These scripts are aimed at initializing control graphs depending on the application:

- type of control variable (velocity, acceleration, torque)
- type of solver (equality only inequality and equality).

## 2.7 Task-Space Inverse Dynamics (TSID)

It is a popular control framework for humanoid robots. It is a C++ library for optimization-based inverse dynamics control. TSID is based on the rigid multi-body dynamics library that is also developed on the Stack of Tasks: Pinocchio.

## **2.8 Sot-dynamic-pinocchio**

This software provides robot dynamic computation for dynamic-graph by using Pinocchio. It is dependent on dynamic-graph, sot-core and Pinocchio.

## **2.9 Sot-pattern-generator**

This software provides jrl-walkgen bindings for the dynamic-graph package. It allows the computation of whole-body biped walk trajectories.

## **2.10 Sot-torque-control**

Collection of dynamic-graph entities aimed at implementing torque control on different robots. It is dependent on dynamic-graph, dynamic-graph-python, sot-core and Pinocchio.

## **2.11 Sot-talos**

This package provides a generic Stack Of Tasks library for the humanoid robot Talos. This library is highly portable and can be used in various simulators, and the robot itself.

## **2.12 Sot-tiago**

This package provides a generic Stack Of Tasks library for the robot Tiago. This library is highly portable and can be used in various simulators, and the robot itself.



## 3 ROS ecosystem

ROS ecosystem is the most spread package manager for robotics.

### 3.1 ROS build farm

To support this ecosystem they created the ROS build farm that provides building, binary packages, continuous integration, testing and analysis for all the robot packages like gazebo, moveit, rviz, etc.

It is an open-source build farm that allows creating a copied build farm if there is a need to change the source code or have better control of the packages.

#### 3.1.1 Jenkins

To have a modular and reusable infrastructure Jenkins is used. This is a tool for continuous integration to coordinate all the packages of the build farm.



# Jenkins

The requirement to create the build farm are those three machines :

- Jenkins master: Managing the execution of jobs
- Jenkins slaves: Performing the builds
- Webserver: file hosting

There are two steps to generate a Jenkins job :

1. Generation of administrative jobs
2. Running of special jobs by the Jenkins master to generate the jobs for all the builds

All this generation is done automatically, the only manual interaction is to synchronize build packages into the main repository. It is done once every month. It exists a ROS testing build farm where the developers of packages can see if their package build successfully. This soaking area permits the developers to work on their packages without breaking the official ROS build farm and have a direct feedback of the stability of their packages and if there are any warning.

### 3.1.2 Rosdistro / ROS versions

It is a GitHub repository that lists all the packages of all the distros:

<https://github.com/ros/rosdistro>

It is composed of folders for each ROS distro release with a file named `distribution.yaml` that contains a list of all packages for the distro.

All the packages in those folders can be downloaded with this command :

**\$ sudo apt install ros-<your\_distro>-<name\_of\_the\_package>**

When a package is added to the list of a distro release the Jenkins jobs are automatically generated. The file `index-v4.yaml` that is at the source of the GitHub determine the path to all the folders of distro releases with the status and the version of ROS.

### 3.1.3 CMake

CMake is an open-source software tool for managing the build process of software on multiplatform. It automatically tests and validates the necessary prerequisite to build. Then it determines the dependencies between the different projects to plan a build that is adapted to the platform used.

In a CMake project, a text file needs to be created: `CmakeLists.txt`. It contains the information that CMake needs to build the project on several platforms.

Understanding the package to code the `CMakeLists.txt` is mandatory. It is needed to specify the dependencies, the inclusion of the code, differentiation of platform (Windows, macOS, Linux), and also needed libraries.

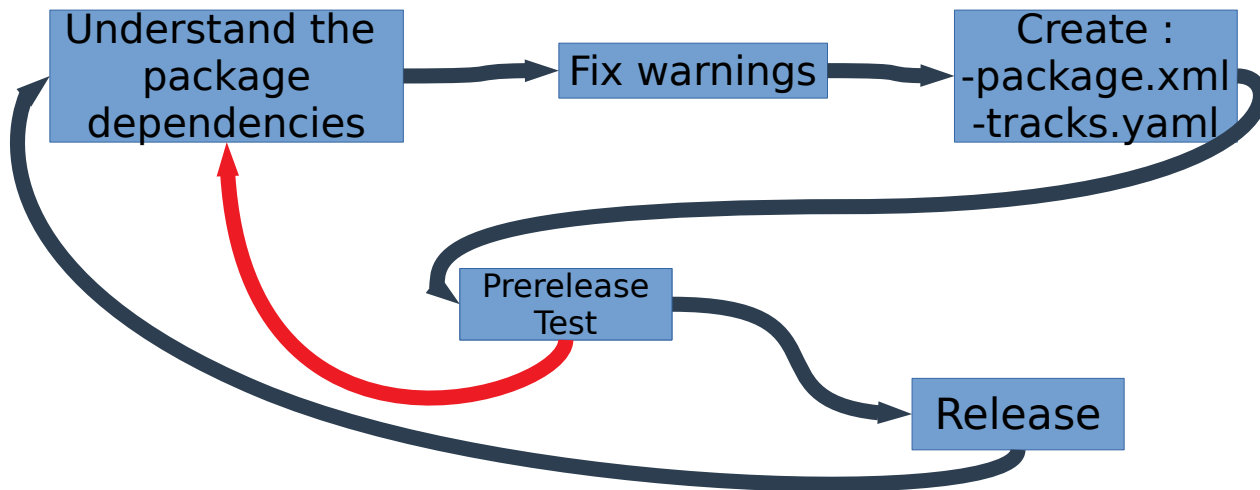
### 3.1.4 Catkin

Catkin is the standard to build packages used in ROS. It is a collection of CMake macros implemented from the ROS Fuerte release. There are several useful commands to simplify the build.

However, the packages that I need to implement are pure Cmake packages and because of that, it cannot use those functions. There is a way to release third party packages that are not using catkin. By creating a `package.xml` we can make ROS use this package as a standard catkin package even if it is not.

## 4 Methodology

Thanks to my new understanding of the SoT I am needed to implement this world of research on ROS so that end users can apply those packages in their projects. When searching for a method to release packages I converged on a methodology to efficiently release those packages on the build farm.



### 4.1 Understand the package dependencies

It exists dependencies for the build, the execution, the tests, and the documentation generation. The packages I have to implement are also interdependent so I could not release all the packages at the same time. I have to do it incrementally.

Build	Execution	Tests	Documentation	Packages
Git	ROS1:Catkin	gtest	doxygen	dynamic-graph
Doxygen	ROS2: ament_cmake			dynamic-graph-python
	eigen			dynamic-graph-tutorial
	boost			sot-core
	graphviz			
	roscpp			
	pinocchio			

Because the packages are pure CMake packages there is the dependency of the buildtool that need to be Cmake. As the Sot repositories are pure CMake packages, it is necessary to specify CMake in the buildtool dependencies.

## 4.2 Fix warnings

Packages	Warnings	Solutions
dynamic-graph	<b>unused variable 'aint' [-Wunused-variable] int aint(anet); ^~~~~</b>	<p>Those errors were due to tests of existence. The variables were just used for the conditions and not in the code. To fix those issues I used those variables in the code :</p> <pre>aint++; abool=!abool; adouble++; aint++; afloat++;</pre> <p>This solution is not elegant but it silenced the warnings and the test are still working.</p>
	<b>unused variable 'abool' [-Wunused-variable] bool abool(anet); ^~~~~~</b>	
	<b>unused variable 'aint' [-Wunused-variable] unsigned int aint(anet); ^~~~~</b>	
	<b>unused variable 'adouble' [-Wunused-variable] double adouble(anet); ^~~~~~</b>	
	<b>unused variable 'afloat' [-Wunused-variable] float afloat(anet); ^~~~~~</b>	
	<b>variable 'res' set but not used [-Wunused-but-set-variable] bool res = false; ^~~</b>	
dynamic-	<p><b>dynamic-graph-python/_build/doc/sphinx/index.rst:18: WARNING: autodoc: failed to import class u'Entity' from module u'dynamic_graph.entity';</b></p> <p><b>dynamic-graph-python/_build/doc/sphinx/index.rst:25: WARNING: autodoc: failed to import class u'SignalBase' from module</b></p>	<p>The build is successful even though there are 4 warnings. We can see that those warnings are linked to Sphinx. It is an automated documentation generator. After discussion with the team, it appears that they were not using it and therefore was not needed. I deleted all</p>

graph-python	<p>u'dynamic_graph.signal_base'</p> <p>dynamic-graph-python/_build/doc/sphinx/index.rst:30: WARNING: autodoc: failed to import module u'dynamic_graph';</p> <p>copying static files... WARNING: html_static_path entry u' dynamic-graph-python/_build/doc/sphinx/_static' does not exist</p> <p>build succeeded, 4 warnings</p>	traces of sphinx in the CmakeList.txt and the GitHub repository.
	Issues to release on ROS foxy and eloquent due to roscpp package that is not working with ROS2 yet.	Removed roscpp from the package.xml
dynamic-graph-tutorial	It was the same errors on sphinx.	I applied the same solution.
TSID	<p>/tsid/tests/tasks.cpp:312:34: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]</p> <p>BOOST_CHECK(constraint.rows()==robot.nv());</p> <p>/tsid/tests/tasks.cpp: In member function 'void tasksTest::test_task_joint_posVelAcc_bounds::test_method()':</p> <p>/home/rascof/tsid/tests/tasks.cpp:364:34: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]</p> <p>BOOST_CHECK(constraint.rows()==robot.na());</p>	The program is comparing signed and unsigned integers; this warning could lead to issues. To fix this, I needed to choose between signed int and unsigned int for the variables before the comparison. I put signed int after discussion with the team because sometime those variables could be equal to -1 to highlight an issue.
	Eiquadprog dependency is not found	Eiquadprog is not on ROS. A solution could be to add it on the ROS build farm or to remove the dependency only on ROS (in discussion)

sot-core	<p>It was the same errors on sphinx.</p> <p>In file included from sot-core/tests/features/test_feature_generic.c pp:34:0:</p> <p>sot-core/include/sot/core/feature- pose.hh:175:39: warning: type attributes ignored after type is already defined [- Wattributes] extern template class SOT_CORE_DLLAPI FeaturePose&lt;SE3Representation&gt;;</p> <p>sot-core/include/sot/core/feature- pose.hh:176:39: warning: type attributes ignored after type is already defined [- Wattributes] extern template class SOT_CORE_DLLAPI FeaturePose&lt;R3xSO3Representation&gt;;</p> <p>In file included from sot-core/include/sot/core/ parameter-server.hh:42:0,  from sot-core/src/tools/parameter- server.cpp:39:</p> <p>sot-core/include/sot/core/robot-utils.hh: In member function 'Type dynamicgraph::sot::RobotUtil::get_parameter (const string&amp;) [with Type = bool]':</p> <p>sot-core/include/sot/core/robot-utils.hh:309:3: warning: control reaches end of non-void function [-Wreturn-type]</p> <p>sot-core/include/sot/core/robot-utils.hh: In member function 'Type dynamicgraph::sot::RobotUtil::get_parameter (const string&amp;) [with Type = double]':</p> <p>sot-core/include/sot/core/robot-utils.hh:309:3: warning: control reaches end of non-void function [-Wreturn-type]</p> <p>sot-core/include/sot/core/robot-utils.hh: In member function 'Type</p>	<p>I applied the same solution.</p> <p>Those warning are very complex because they are due to the fact that I compile with a compiler that doesn't have the understanding of those 2 lines. I don't have the knowledge to fix this issue because it is a new functionality developed on version superior to c++ 2011.</p> <p>Add a return 0; at the end of the loop.</p>
----------	---	--

	<p>dynamicgraph::sot::RobotUtil::get_parameter (const string&amp;) [with Type = int]':</p> <p>/home/rascof/sot-core/include/sot/core/robot-utils.hh:309:3: warning: control reaches end of non-void function [-Wreturn-type]</p> <p>sot-core/include/sot/core/robot-utils.hh: In member function 'Type dynamicgraph::sot::RobotUtil::get_parameter (const string&amp;) [with Type = std::__cxx11::basic_string&lt;char&gt;]':</p> <p>sot-core/include/sot/core/robot-utils.hh:309:3: warning: control reaches end of non-void function [-Wreturn-type]</p>	
	<p>In file included from sot-core/src/tools/parameter-server.cpp:39:0:</p> <p>sot-core/include/sot/core/parameter-server.hh: In member function 'Type dynamicgraph::sot::ParameterServer::getParameter(const string&amp;) [with Type = bool]':</p> <p>sot-core/include/sot/core/parameter-server.hh:122:14: warning: 'ParameterValue' may be used uninitialized in this function [-Wmaybe-uninitialized] return ParameterValue;</p> <p>sot-core/include/sot/core/parameter-server.hh: In member function 'Type dynamicgraph::sot::ParameterServer::getParameter(const string&amp;) [with Type = double]':</p> <p>sot-core/include/sot/core/parameter-server.hh:122:14: warning: 'ParameterValue' may be used uninitialized in this function [-Wmaybe-uninitialized] return ParameterValue;</p> <p>/home/rascof/sot-core/include/sot/core/parameter-server.hh: In member function 'Type dynamicgraph::sot::ParameterServer::getParameter(const string&amp;) [with Type = int]':</p>	<p>Initialize the generic variable Type with the code :</p> <p>Type ParameterValue=Type();</p>

	<b>sot-core/include/sot/core/parameter-server.hh:122:14: warning: 'ParameterValue' may be used uninitialized in this function [-Wmaybe-uninitialized] return ParameterValue;</b>	
sot-tools	<b>Issues to release on ROS foxy and eloquent due to roscpp package that is not working with ROS2 yet.</b>	<p>Dependency on Roscpp for ROS1 and rclcpp for ROS2</p> <p>Some code need to be adapted to match the requirment of rclcpp</p>



## 4.3 Create the files

### 4.3.1 Package.xml

The creation of the package.xml in the upstream repository is codified with a lot of option to add different types of dependencies.

Before the modification the original package.xml in the stack of tasks for dynamic-graph looks like this :

```
1 <package format="2">
2   <name>dynamic-graph</name>
3   <version>4.1.0</version>
4   <description>
5     Dynamic graph library
6   </description>
7   <maintainer email="ostasse@laas.fr">Olivier Stasse</maintainer>
8   <license>BSD</license>
9
10  <url>http://github.com/stack-of-tasks/dynamic-graph</url>
11  <author>Nicolas Mansard</author>
12  <author>Olivier Stasse</author>
13
14  <buildtool_depend>catkin</buildtool_depend>
15
16  <doc_depend>doxygen</doc_depend>
17
18 </package>
```

The specifications for releasing a third-party package are as below :

- Have a package.xml in the source of the upstream repository on GitHub
- Have a <build\_type> tag on CMake in the <export> tag at the end of the file
- Add an install rule in the CmakeList.txt file
- Have an <exec\_depend> tag on catkin in the file for ROS1 and an <exec\_depend> on ament\_cmake for ROS2

For the packages in SoT it is implemented like this :

```

<?xml version="1.0"?>
<package format="3">
  <name>dynamic-graph</name>
  <version>4.2.1</version>
  <description>
    Dynamic graph library
  </description>
  <maintainer email="ostasse@laas.fr">Olivier Stasse</maintainer>
  <license>BSD</license>

  <url>http://github.com/stack-of-tasks/dynamic-graph</url>
  <author>Nicolas Mansard</author>
  <author>Olivier Stasse</author>

  <build_depend>git</build_depend>
  <build_depend>doxygen</build_depend>
  <!-- The following tags are recommended by REP-136 -->
  <exec_depend condition="$ROS_VERSION == 1">catkin</exec_depend>
  <exec_depend condition="$ROS_VERSION == 2">ament_cmake</exec_depend>
  <depend>eigen</depend>
  <depend>boost</depend>
  <depend>graphviz</depend>

  <buildtool_depend>cmake</buildtool_depend>

  <export>
    <build_type>cmake</build_type>
  </export>
</package>

```

// format 3 to be able to make a difference between ROS1 and ROS2

//exec depend on catkin for ROS1

//exec\_depend on ament\_cmake for ROS2

//export tag with CMake

And at the end of the **CMakeLists.txt** :

```
INSTALL(FILES package.xml DESTINATION share/${PROJECT_NAME})
```

Generally, to add a new dependency there is just a need to put a depend on the tag. For example, if a git dependency is missing a new line can be added:

<depend>git</depend> to fix the dependency.

The <depend> tag is equivalent to <build\_depend>,<exec\_depend> and <test\_depend> tag on a module.

To match the previously seen dependencies I added eigen, boost and graphviz (that contain the dot module that was needed to print the graph in the dynamic-graph package).

### 4.3.2 Tracks.yaml

Tracks.yaml is an important file since it helps to create branches of the project during the release so that the package can be built on different platforms and ROS versions.

A typical tracks.yaml file looks like this:

```
tracks:
  eloquent:
    actions:
      - bloom-export-upstream :{vcs_local_uri} :{vcs_type} --tag :{release_tag} --display-uri
        :{vcs_uri} --name :{name} --output-dir :{archive_dir_path}
      - git-bloom-import-upstream :{archive_path} :{patches} --release-version :{version}
        --replace
      - git-bloom-generate -y rosrelease :{ros_distro} --source upstream -i :{release_inc}
      - git-bloom-generate -y rosdebian --prefix release/:{ros_distro} :{ros_distro}
        -i :{release_inc} --os-name ubuntu
      - git-bloom-generate -y rosdebian --prefix release/:{ros_distro} :{ros_distro}
        -i :{release_inc} --os-name debian --os-not-required
      - git-bloom-generate -y rosrpm --prefix release/:{ros_distro} :{ros_distro} -i
        :{release_inc}
    devel_branch: devel
    last_release: v4.2.2
    last_version: 4.2.2
    name: dynamic-graph
    patches: null
    release_inc: '1'
    release_repo_url: https://github.com/stack-of-tasks/dynamic-graph-ros-release.git
    release_tag: v:{version}
    ros_distro: eloquent
    vcs_type: git
    vcs_uri: https://github.com/stack-of-tasks/dynamic-graph.git
    version: :{auto}
  foxy:
    actions:
      - bloom-export-upstream :{vcs_local_uri} :{vcs_type} --tag :{release_tag} --display-uri
        :{vcs_uri} --name :{name} --output-dir :{archive_dir_path}
```

To add other ROS version, the same syntax is needed with only changing the ROS version name and the **ros\_distro**.

The actions are always the same between the packages; they are used by bloom to create the branches and add files to build on different platforms.

**devel\_branch** is the branch of your upstream repository where you actively develop your code.

**last\_release** and **last\_version** are the last release version of the code on Robotpkg. Here the user needs to be cautious when using the notation “v”

because it needs to be specified on the **release\_tag** so that the **version** could be set to auto.

**release\_inc** is the number of releases of this version made by the user on the build farm.

**release\_repo\_url** is the URL of the GitHub repository where the tracks.yaml file is located.

**ros\_distro** is the ROS version of the package to be released in. So there are 4 versions on the tracks.yaml file: noetic (ROS1), melodic(ROS1), foxy (ROS2) and eloquent(ROS2).

**vcs\_type** is the version control system type used for the package. Here since the packages are on GitHub the version control type is git.

**vcs\_uri** is the URL of the GitHub upstream repository where the developed code is located. Tracks.yaml is an important file since it helps to create branches of the project during the release so that the package can be built on different platforms and ROS versions.

## 4.4 Prerelease with Docker

To not release broken packages into the build farm, it is possible to do a prerelease by generating Jenkins jobs just like the build farm locally. If the user releases a broken package into the build farm; it will try to build this package every 15 minutes and use Jenkins slaves for nothing. The user will be asked to remove the package from the `distribution.yaml`.

ROS maintainers created a site to generate a prerelease python program. The user needs to specify the package he wants to release. It is even possible to put a custom package that is not yet on the ROS build farm. This prerelease highlight build issues and missing dependencies. Local files can be directly modified to fix those issues before putting it into the project GitHub repository.

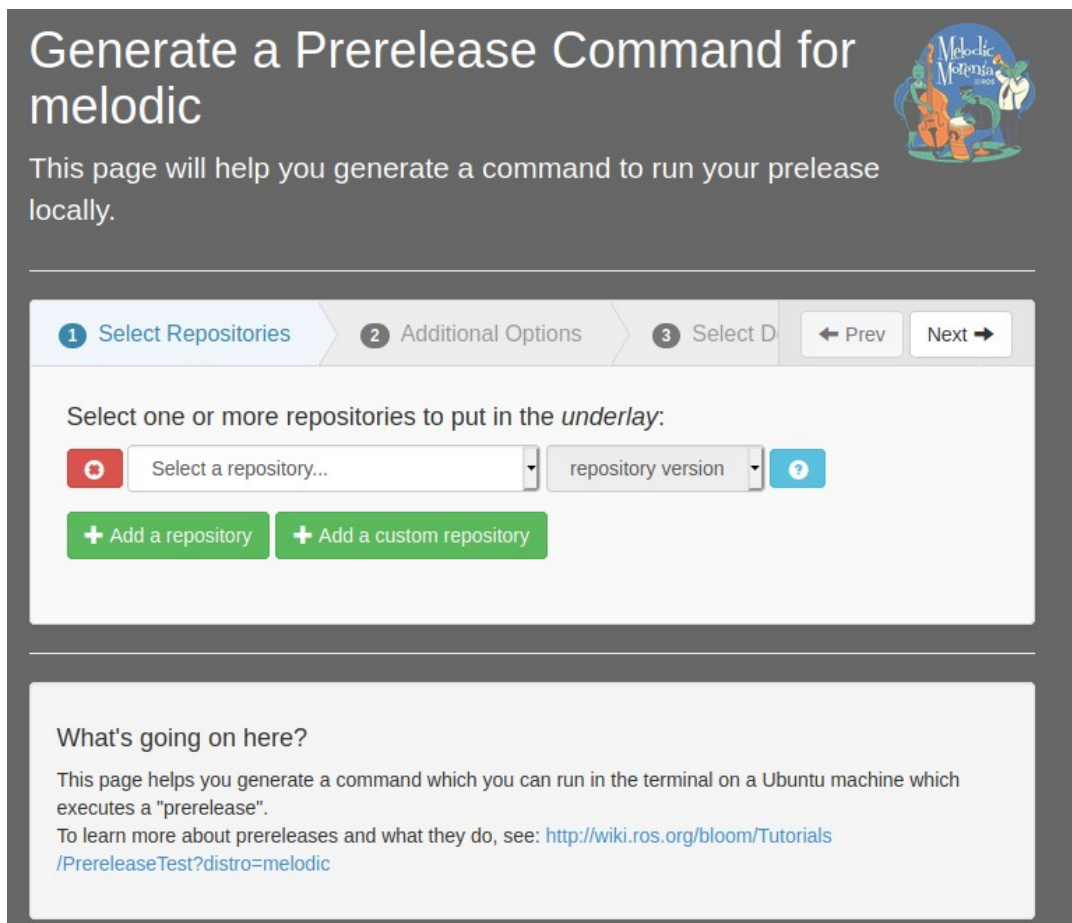


Figure 1-10 Interface of the website

This prerelease is performed on a docker. Docker is a simulation of a Linux system where the desired packages can be downloaded. The link can be sent to others to access the docker to have the same configuration. It is useful for

example when the user needs to create a tutorial remotely and everyone needs to have the needed packages at the right version.

This docker download the ROS distro with the packages that are specified in the package.xml file, three distros (version) are available on the site and it is not yet created for ROS2 :



**Figure 1-11 ROS distro available on the website for prerelease**

The time it takes to do the prerelease depends on the number of test packages and the number of dependencies to download.

## 4.5 Release

When everything is checked it is possible to proceed with the release. Make sure that all the errors are fixed because if some are left the release will be unstable and there will be a need to patch the warnings and proceed again with the whole process.

## 5 Tutorial

It exists an official tutorial to do the releases:

<https://wiki.ros.org/bloom/Tutorials/ReleaseThirdParty>

I inspired myself from this tutorial and adapted it to the laboratory standard.

### 5.1 Release for the first time

In the official tutorial, the first chapters are on using catkin command to prepare the release. Because our packages are not developed with catkin the release need to be done as usual by the person in charge of releasing packages for the CNRS.

The command to do a release is straight forward however it needs some preparations to work. For this tutorial, we assume that the package.xml is well created with all the good dependencies.

#### 5.1.1 Github release repository

First, you will need to create a GitHub repository with the name :

<your\_package>-release

You need to check the case: **Initialize this repository with a README** to create the first commit.

#### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*      Repository name \*  
Rascof / tsid-release ✓

Great repository names are short and memorable. Need inspiration? How about [studious-parakeet?](#)

Description (optional)

**Public**  
Anyone on the internet can see this repository. You choose who can commit.

**Private**  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

**Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾      Add a license: None ▾ ⓘ

Create repository

**Figure 1-12 Creation of a GitHub repository for the release of TSID**

Then you will have to create a tracks.yaml file initiated with the distros you want to release your package on. Like this :

```
tracks:
  eloquent:
    actions:
      - bloom-export-upstream :{vcs_local_uri} :{vcs_type} --tag :{release_tag} --display-uri
        :{vcs_uri} --name :{name} --output-dir :{archive_dir_path}
      - git-bloom-import-upstream :{archive_path} :{patches} --release-version :{version}
        --replace
      - git-bloom-generate -y rosrelease :{ros_distro} --source upstream -i :{release_inc}
      - git-bloom-generate -y rosdebian --prefix release/{ros_distro} :{ros_distro}
        -i :{release_inc} --os-name ubuntu
      - git-bloom-generate -y rosdebian --prefix release/{ros_distro} :{ros_distro}
        -i :{release_inc} --os-name debian --os-not-required
      - git-bloom-generate -y rosrpm --prefix release/{ros_distro} :{ros_distro} -i
        :{release_inc}
    devel_branch: devel
    last_release: v4.2.2
    last_version: 4.2.2
    name: dynamic-graph
    patches: null
    release_inc: '1'
    release_repo_url: https://github.com/stack-of-tasks/dynamic-graph-ros-release.git
    release_tag: v:{version}
    ros_distro: eloquent
    vcs_type: git
    vcs_uri: https://github.com/stack-of-tasks/dynamic-graph.git
    version: :{auto}
  foxy:
    actions:
      - bloom-export-upstream :{vcs_local_uri} :{vcs_type} --tag :{release_tag} --display-uri
        :{vcs_uri} --name :{name} --output-dir :{archive_dir_path}
```

It is possible to directly copy/paste the tracks.yaml file from any packages. Here is a link with a well-created track.yaml:

<https://github.com/stack-of-tasks/dynamic-graph-ros-release/blob/master/tracks.yaml>

The few changes that need to be performed are the **last\_release**, **last\_version** It needs to be to the last release of the package (do not forget the "v" before the version since it is a standard from the LAAS).

The name also needs to be updated.

The URL of the release repository that was just created (**release\_repo\_url**) and the URL of the upstream repository where the code of the package is developed (**vcs\_uri**) need to be changed appropriately.



If the reader wants more information on this file it is possible to refer to this site <https://wiki.ros.org/bloom/Tutorials/FirstTimeRelease> at section 4.1 configure a Release Track.

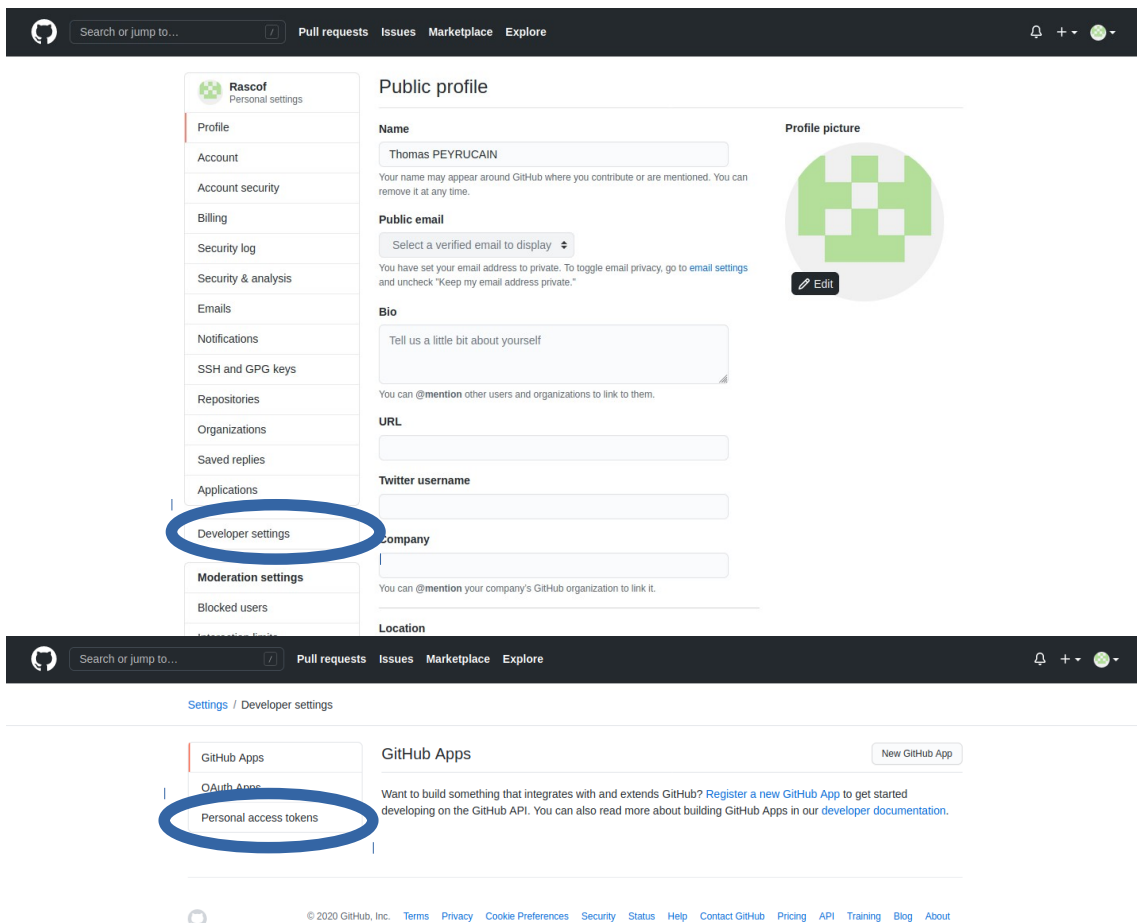
### 5.1.2 Authorizing Bloom to generate pull request on the rosdistro repository

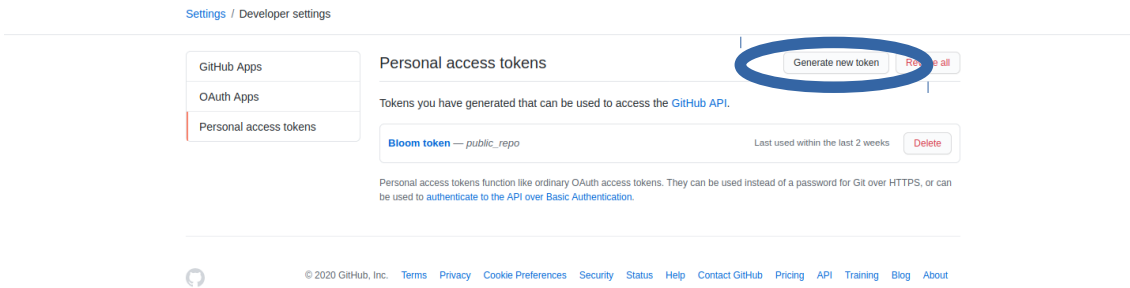
Make sure to have a file `~/.config/bloom` well initiated :

```
{  
  "github_user": "<your-github-username>",  
  "oauth_token": "<token-you-created-for-bloom>"  
}
```

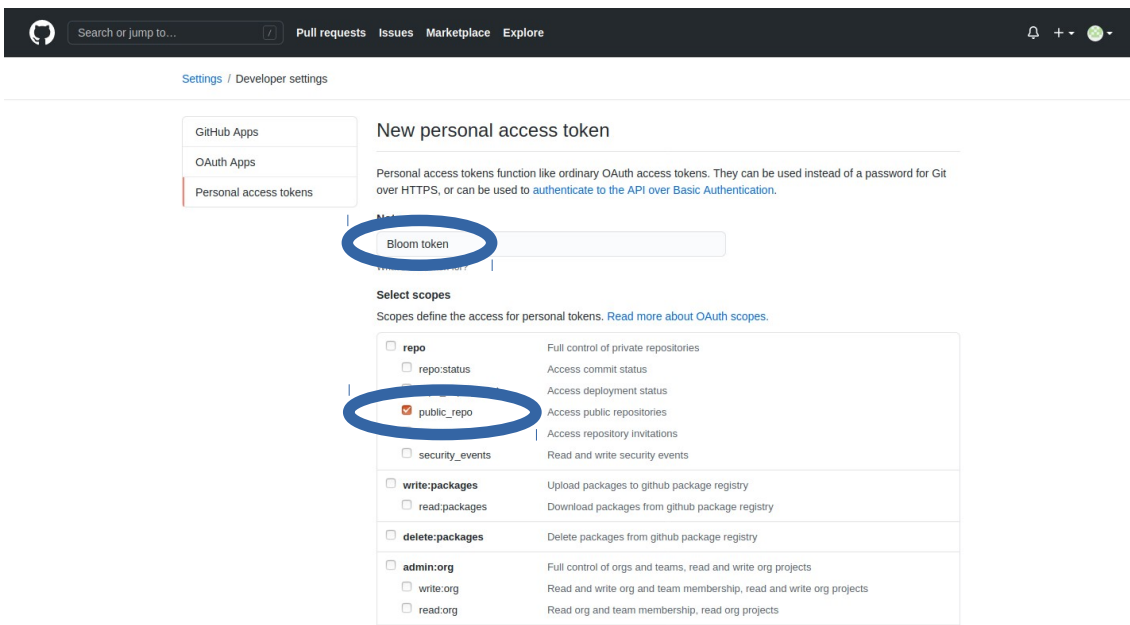
If not created you need to create it. Then log in to your GitHub account and generate a new token with “public\_repo” granted :

- Go into settings and Developer settings and then personal access tokens

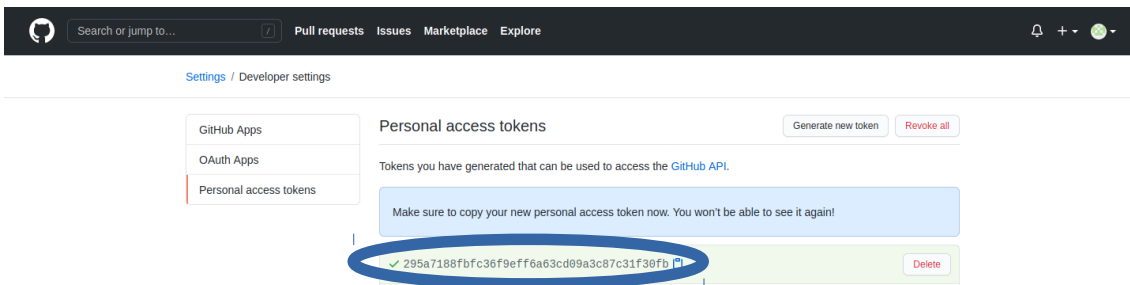




Then enter your password when asked



Then generate the token with public\_repo ticked



Copy this code inside the bloom file. For example:

```
{
  "github_user": "Rascof",
  "oauth_token": "295a7188fbfc36f9eff6a63cd09a3c87c31f30fb"
}
```

### 5.1.3 Release

Release command :

```
bloom-release --rosdistro <ros_distro> --track <ros_distro> <package_name>
```

For example with TSID package:

```
rascof@rascof-Lenovo-Y520-15IKBN:~$ bloom-release --rosdistro melodic --track melodic tsid
Specified repository 'tsid' is not in the distribution file located at 'https://raw.githubusercontent.com/ros/rosdistro/master/melodic/distribution.yaml'
Could not determine release repository url for repository 'tsid' of distro 'melodic'
You can continue the release process by manually specifying the location of the RELEASE repository.
To be clear this is the url of the RELEASE repository not the upstream repository.
For release repositories on GitHub, you should provide the `https://` url which should end in `.git`.
Here is the url for a typical release repository on GitHub: https://github.com/ros-gbp/rviz-release.git
it
==> Looking for a release of this repository in a different distribution...
No reasonable default release repository url could be determined from previous releases.
Release repository url [press enter to abort]:
```

Then it is necessary to enter the GitHub repository that was just created with .git at the end :

[https://github.com/<repository>/<your\\_package\\_name>-release.git](https://github.com/<repository>/<your_package_name>-release.git)

You will have to enter your GitHub name and password. Make sure to have the administrator right to modify the release repository.

The error message is okay because it is the first release on the build farm; the package is not yet on the ROS distro distribution.yaml.

```
Everything went as expected, you should check that the new tags match your expectations, and then push to the release repo with:
  git push --all && git push --tags # You might have to add --force to the second command if you are over-writing existing tags
<== Released 'dynamic-graph-python' using release track 'foxy' successfully
==> git remote -v
origin https://github.com/stack-of-tasks/dynamic-graph-python-ros-release.git (fetch)
origin https://github.com/stack-of-tasks/dynamic-graph-python-ros-release.git (push)
Releasing complete, push to release repository?
Continue [Y/n]?
```

Until there no modification was made to the release repository. Enter “y” if you are sure of your release.

Enter your GitHub name and password with caution since if you misspell your name or password the release will fail and you will have to start all over again. You will be asked two times. One for creating the branches on your upstream repository and one for adding the tags of the version of your release.

It is asking for some more information for the distribution.yaml:

- Documentation information: **Enter “y”**
  1. VCS type: **git**
  2. VCS url: **your upstream repository were you develop your code**
  3. VCS version: **the branch where the code is developed. Most of the time it is devel for the packages from the SOT**
- Source information: **Enter “y”**
  1. VCS type: **git**
  2. VCS url: **your upstream repository were you develop your code**
  3. VCS version: **the branch where the code is developed. Most of the time it is devel for the packages from the SOT**
- Turn on pull request testing: **Enter “y”**
- Maintenance status: **Enter “y”**
  - You have 4 choices: developed, maintained, unmaintained, end-of-life. Most of the time it will be **maintained** since packages from the LAAS are still in development to fix issues on packages and improve them.
- Press **Enter** if you put another maintenance status that end-of-life

If everything works well the program should ask you to do a push request to the distribution.yaml file if not go into the issue section to see if the error is known.

```

Status Description [press Enter for no change]:
Unified diff for the ROS distro file located at '/tmp/tmpSu817d/dynamic-graph-ros-release-4.2.1-2.patch':
--- melodic/distribution.yaml
+++ melodic/distribution.yaml
@@ -2094,6 +2094,23 @@
     release: release/melodic/{package}/{version}
     url: https://github.com/Achille/dual_quaternions_ros-release.git
     version: 0.1.3-1
+   status: maintained
+ dynamic-graph-ros-release:
+   doc:
+     type: git
+     url: https://github.com/stack-of-tasks/dynamic-graph.git
+     version: devel
+   release:
+   packages:
+   - dynamic-graph
+   tags:
+     release: release/melodic/{package}/{version}
+     url: https://github.com/stack-of-tasks/dynamic-graph-ros-release.git
+     version: 4.2.1-2
+   source:
+     type: git
+     url: https://github.com/stack-of-tasks/dynamic-graph.git
+     version: devel
+     status: maintained
+ dynamic_reconfigure:
+   doc:
==> Checking on GitHub for a fork to make the pull request from...
Could not find a fork of ros/rosdistro on the $Rascof GitHub account.
Would you like to create one now?
Continue [Y/n]? y

```

If this work it should automatically create a pull request to the distribution.yaml of the ROS distro that was chosen. If so go directly to the 5.1.4 section.

However, if it is not working you will see this message:

```

Continue [Y/n]? y
Aborting pull request: HTTP Error 401: Unauthorized (https://api.github.com/repos/ros/rosdistro/forks)
The release of your packages was successful, but the pull request failed.
Please manually open a pull request by editing the file here: 'https://raw.githubusercontent.com/ros/rosdistro/master/melodic/distribution.yaml'
<== No pull request opened.

```

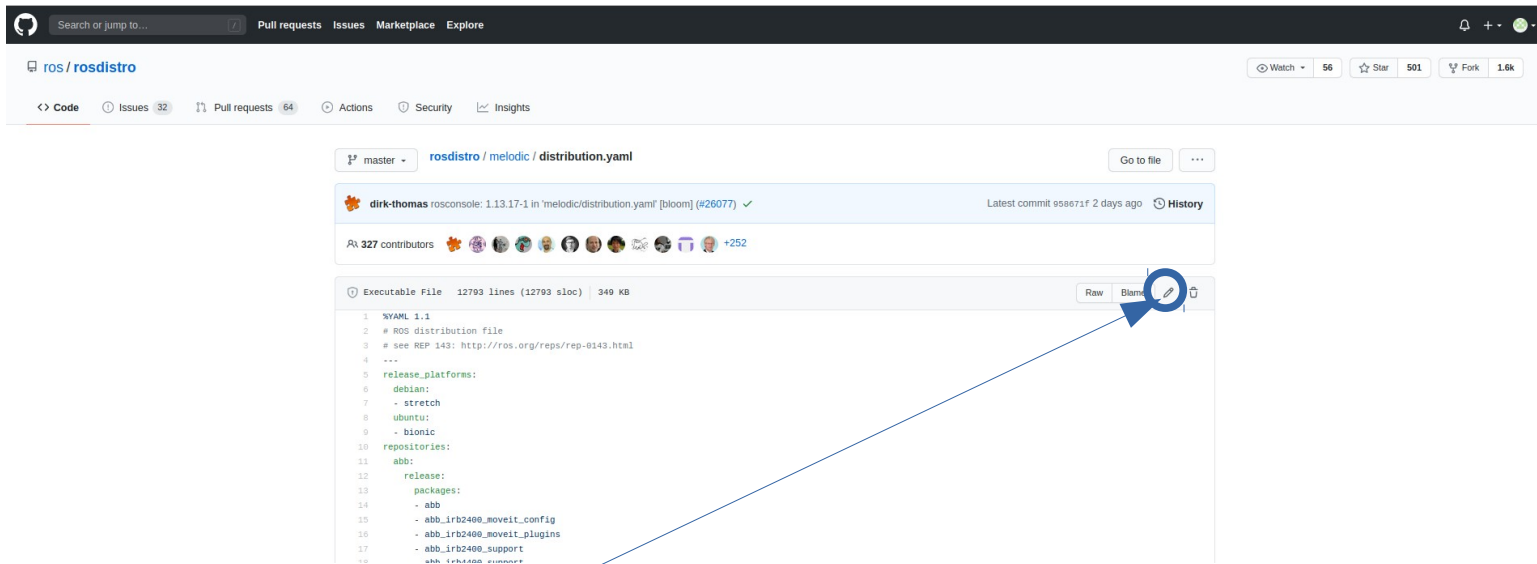
Stay on this window since you will have to access the information created by bloom to add to the distribution.yaml.

So you will have to do the pull request manually

### 5.1.4 Pull request

Go to the official rosdistro GitHub repository where every version of ROS is listed with every available package: <https://github.com/ros/rosdistro>

Click to the folder of the rosdistro that was just released and then on the distribution.yaml file that is inside it.



Click to edit the document

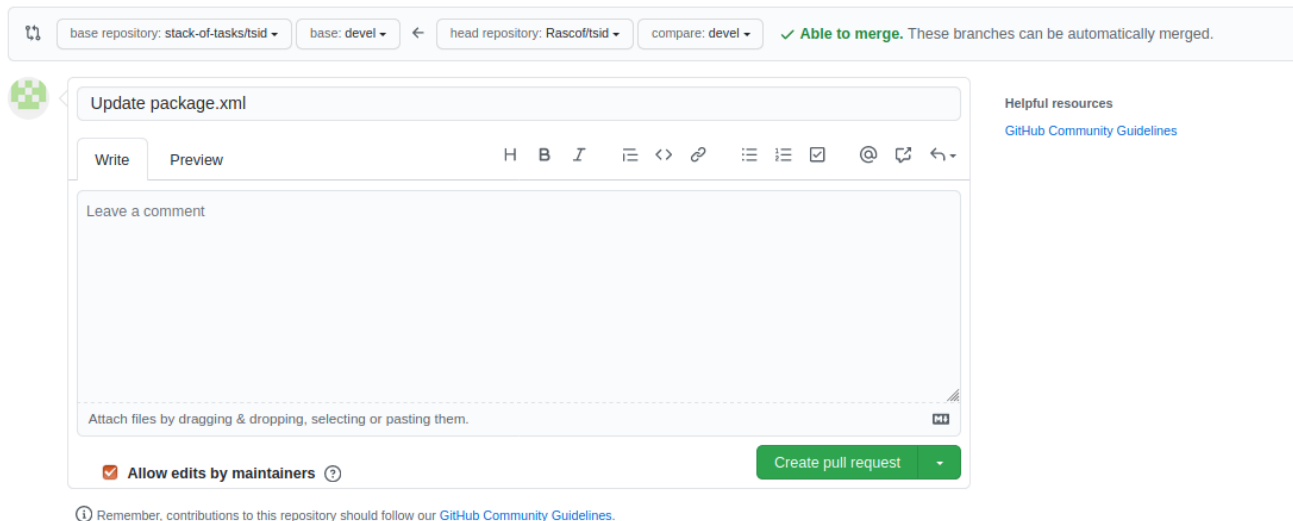
The Ctrl+F to search for the package just after alphabetically here in the example it is **dynamic\_reconfigure**.

Then add the lines that were in the last section at the right place.

Then create the pull request with information about the package.

### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



The pull request is successfully created and a ROS deputy is informed to review the release.

## 5.1.5 Validation of your pull request by a ROS deputy

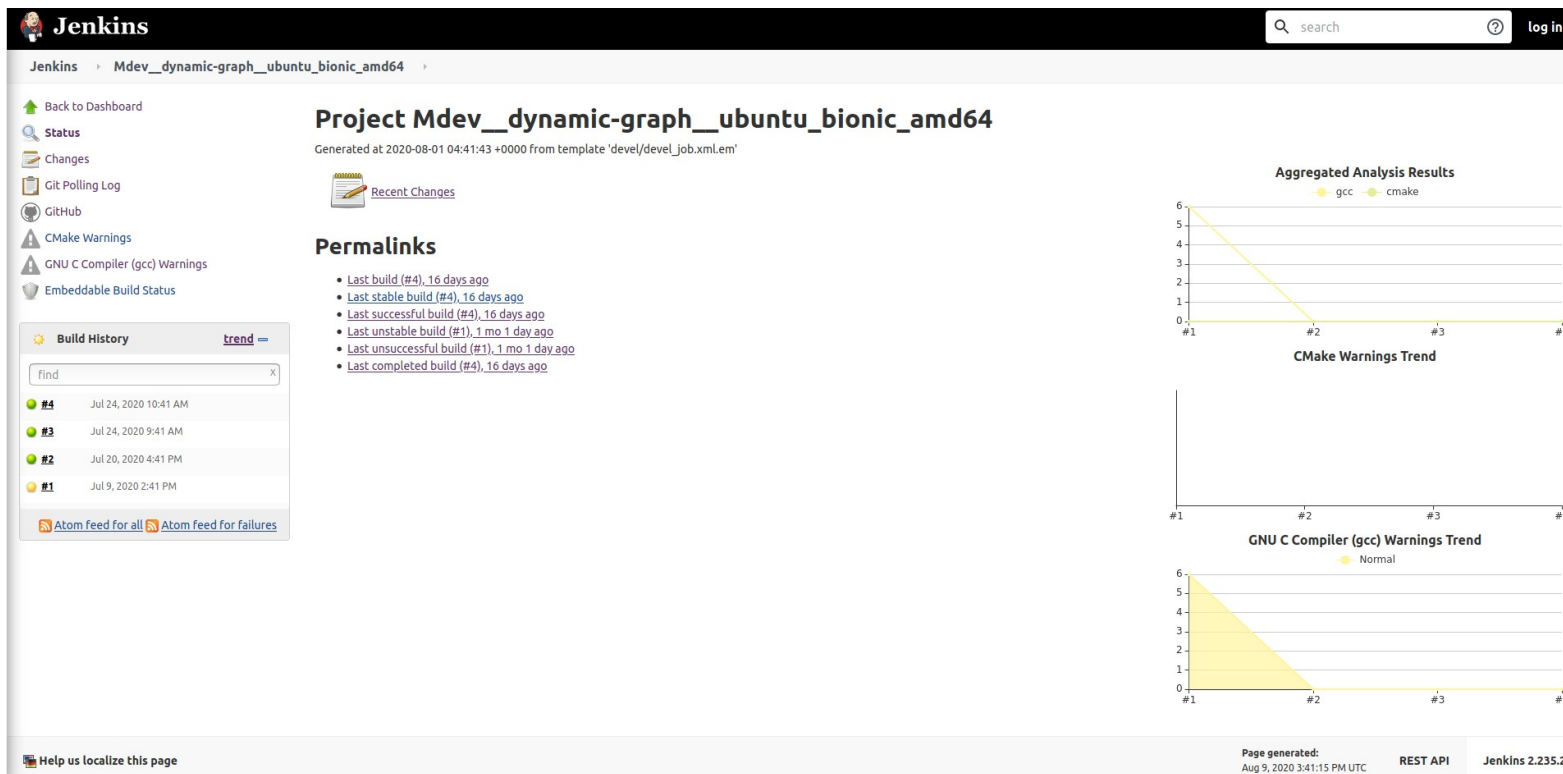
You will have to wait before he makes the review and merges your package into the master branch.

Once it is merged Jenkins will automatically build the package on the build farm and the user will see if the package is stable or not.

If everything went well now your package is on the testing build farm. You can access the test build farm thanks to this tutorial:

<http://wiki.ros.org/action/show/TestingRepository?action=show&redirect=ShadowRepository>

By searching the name of the package on Jenkins it is possible to see the builds performed with warnings and statistics:



Finally, the user will have to wait for a release from the ROS testing build farm to the official ROS build farm. It is done once every 2 to 3 weeks.

Then congratulation the package is on the official ROS build farm !

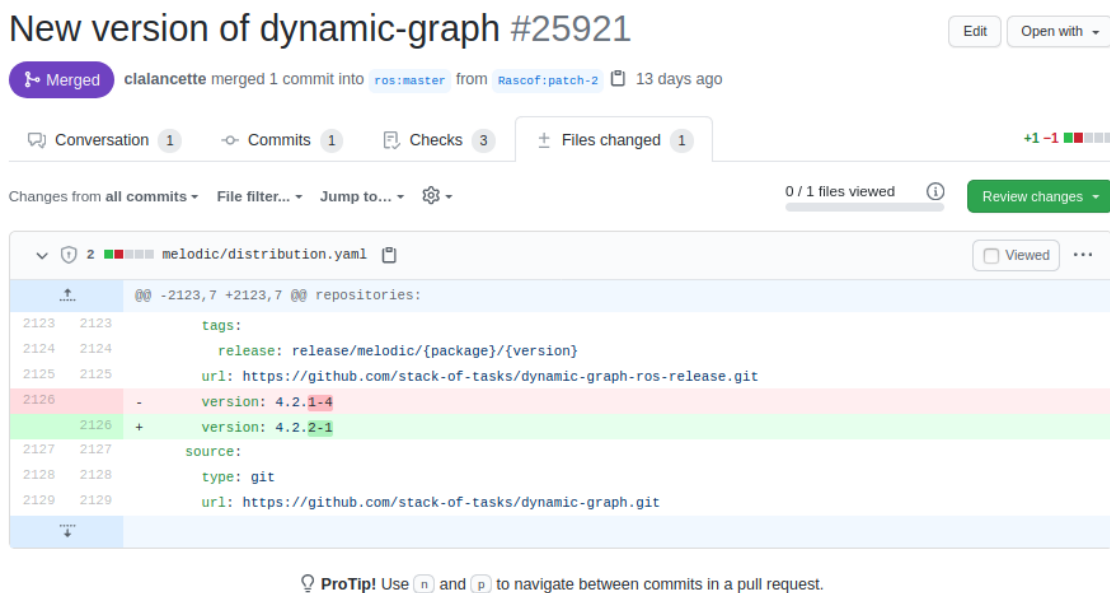
## 5.2 Maintenance of packages

For the maintenance you need only this release command :

```
bloom-release --rosdistro <ros_distro> --track <ros_distro> <package_name>
```

Because the package is already on the build farm the process will continue and the user will be asked his GitHub name and password.

The only modification to do is to update the version number on your package:



The screenshot shows a GitHub pull request titled "New version of dynamic-graph #25921". It is merged and shows changes to the file "melodic/distribution.yaml". The diff highlights the update of the version number from 4.2.1-4 to 4.2.2-1. The diff content is as follows:

```
@@ -2123,7 +2123,7 @@ repositories:
2123 2123     tags:
2124 2124         release: release/melodic/{package}/{version}
2125 2125         url: https://github.com/stack-of-tasks/dynamic-graph-ros-release.git
2126 -     version: 4.2.1-4
2126 +     version: 4.2.2-1
2127 2127     source:
2128 2128         type: git
2129 2129         url: https://github.com/stack-of-tasks/dynamic-graph.git
```

Then the user will have to wait until a ROS deputy validates the new version of your package.

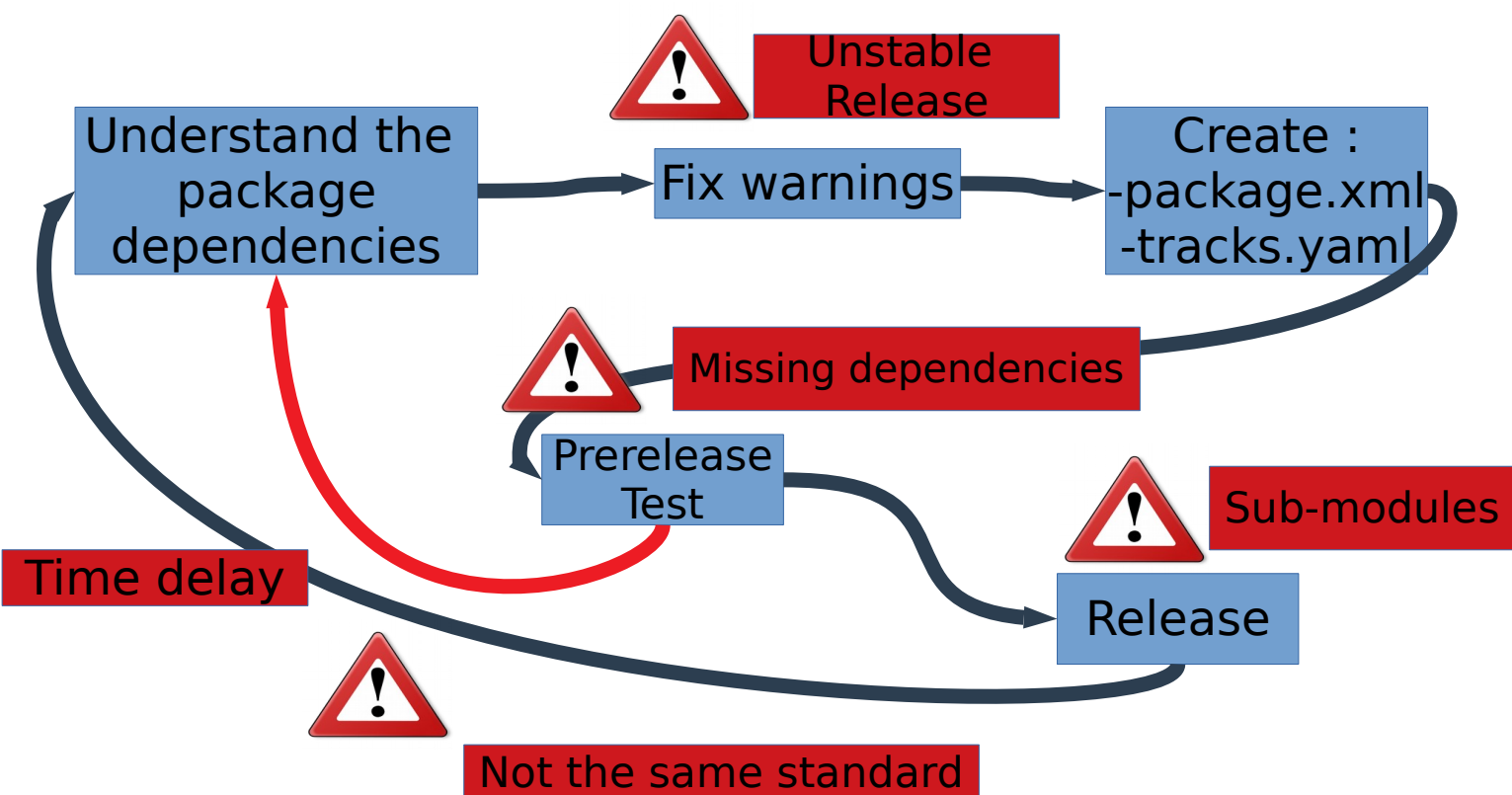
## 5.3 Technical discussion

To create a tutorial that can be adapted to the change of standard I needed to understand the evolution of the standard of the ROS build farm and what I could extract from the literature review. During my research, I saw that the evolution is going in the direction to more third party packages to expand more and more the influence of ROS and the wide range of applications. The standard is still in active maintenance and can change rapidly. There is a need to stay tuned on new versions to watch for incremental change on this site: <https://github.com/ros-infrastructure/rep>

This is mandatory to keep your packages on the build farm.



## 6 Issues



### 6.1 First release of dynamic-graph

#### 6.1.1 Warnings

For the package to be stable on the build farm it needs to be without any warnings. However, it is not the case for the packages released on robotpkg. Developers at the laboratory don't have the time and motivation to fix small issues that don't influence the successful build of packages. I needed to understand the warnings and make changes where it was needed. I discussed those warnings during the meetings and when everything was agreed I made a pull request to put the changes on GitHub.

The warnings were different for each package.

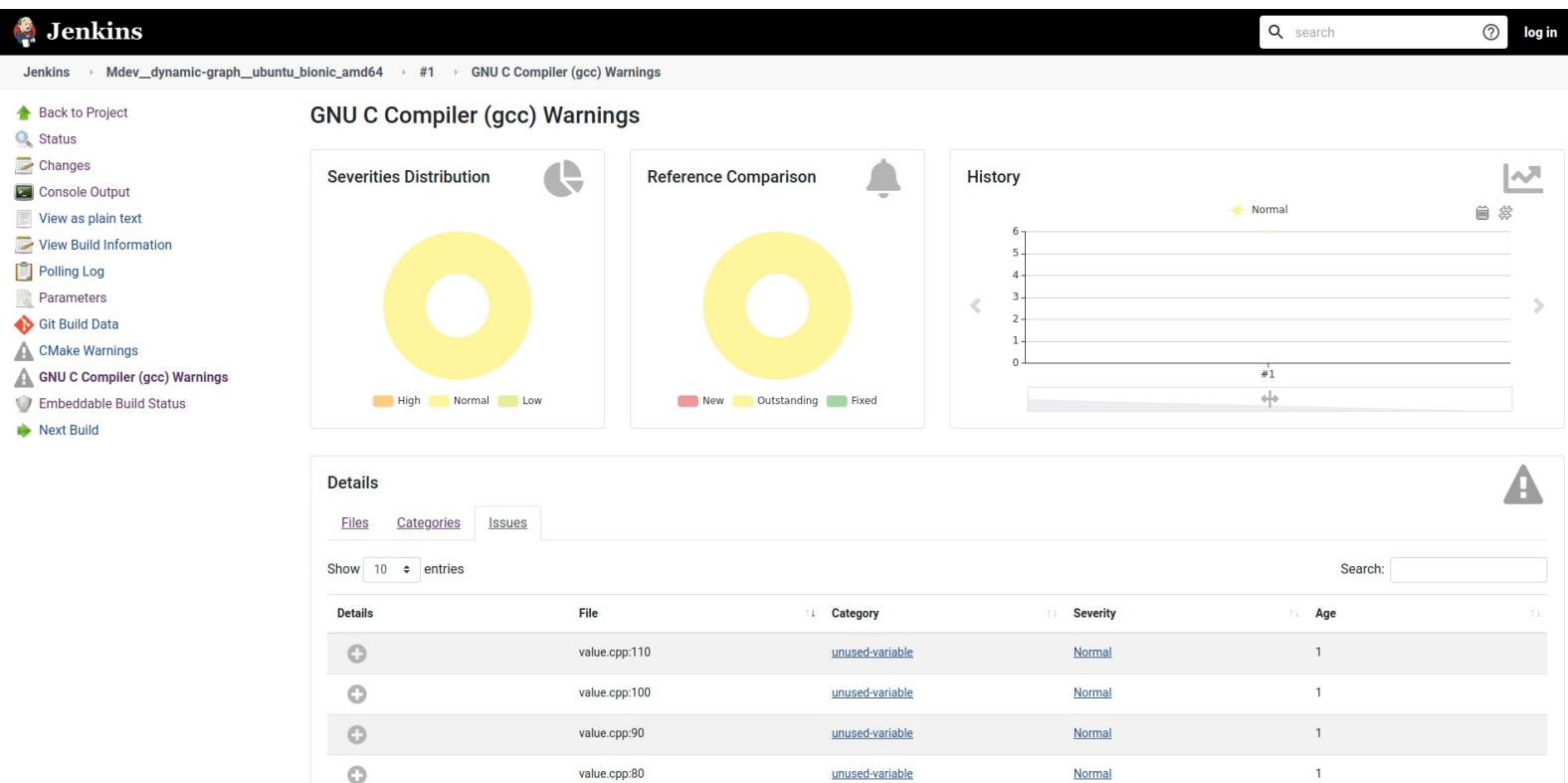


Figure 1-13 Jenkins site that shows the unstable release due to warnings

## 6.1.2 Missing dependencies

If during the analysis of the dependencies I missed some the prerelease process highlight the missing dependencies with error messages.

For example, when doing the prerelease for dynamic-graph I have this error :

**Dot: not found**

The solution to solve this issue is to add Graphviz dependency because it contains the dot module.

So I added `<depend>graphviz</depend>` in package.xml.

## 6.1.3 Submodule issues

When I was testing the release on dynamic-graph for the first time it was not working; I had an error linked to the sub-modules of GitHub.

Submodules are pointing to other GitHub repositories and if the repository is cloned with this option: `--recurse-submodules` it download the submodules with the package.

For instance to clone dynamic-graph with its submodules:

```
git clone --recurse-submodules --branch=devel https://github.com/stack-of-tasks/dynamic-graph
```

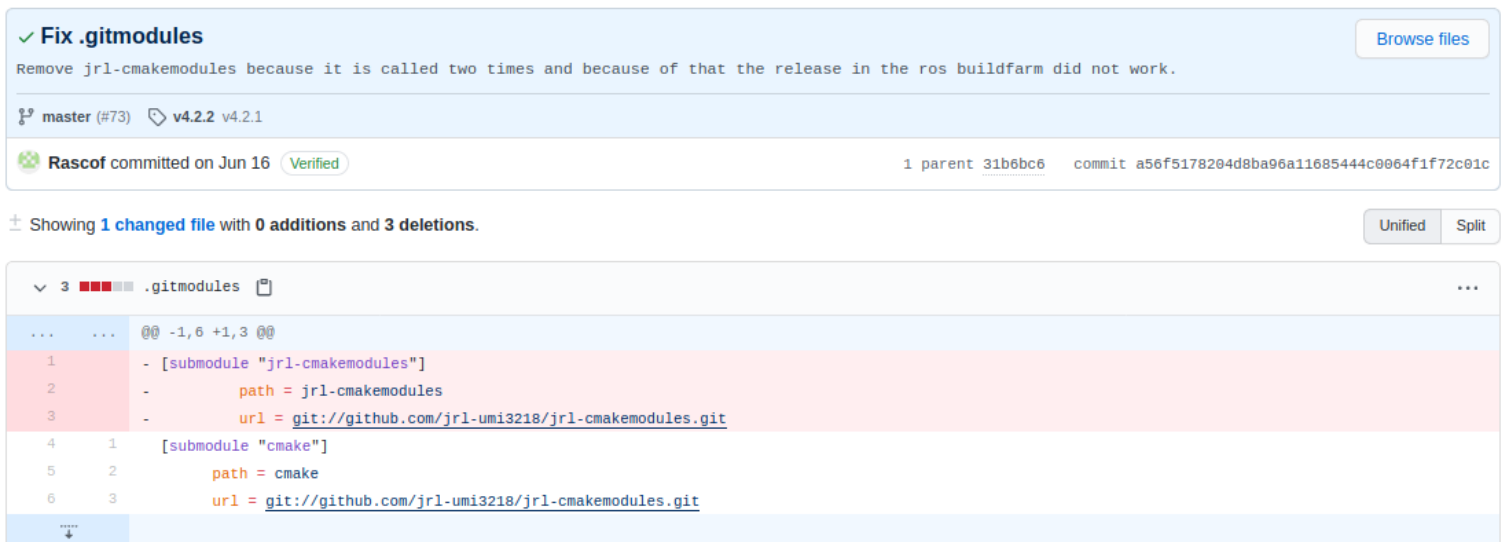
Most of the packages are using submodules to rely on CMake macros developed by the Joint Japanese-French Laboratory. The 2 main submodules used by the laboratory for their packages are a shared CMake submodule and a Travis submodule for continuous integration. Shared CMake submodule simplifies CMake mechanism to have a uniform release of packages. The Travis submodule contains build script to be used by Travis during continuous integration.

To add a submodule the code line is:

**git submodule add <github\_link>**

This command is adding the GitHub link as a submodule and is creating a .gitmodules that list all the modules added.

However, if to remove a submodule, only the local file is removed from the upstream repository it is still listed in the .gitmodules. Because of that, the program is still searching for the submodule even though it has been deleted. That was the error that was occurring. To fix it I removed the three lines that were pointing on an unexisting submodule:



The screenshot shows a GitHub pull request interface. At the top, the title is "Fix .gitmodules" with a green checkmark icon. Below the title, a message reads: "Remove jrl-cmakemodules because it is called two times and because of that the release in the ros buildfarm did not work." To the right of this message is a "Browse files" button. Below the message, the repository path is "master (#73)" and the version is "v4.2.2 v4.2.1". The commit information shows "Rascof committed on Jun 16" with a "Verified" badge. The commit hash is "a56f5178204d8ba96a11685444c0064f1f72c01c" and the parent commit is "31b6bc6". Below this, it says "Showing 1 changed file with 0 additions and 3 deletions." To the right are "Unified" and "Split" view options. The main content area shows a diff for the file ".gitmodules". The diff shows three lines being deleted (indicated by a red background):

```
@@ -1,6 +1,3 @@
1 - [submodule "jrl-cmakemodules"]
2 -     path = jrl-cmakemodules
3 -     url = git://github.com/jrl-umi3218/jrl-cmakemodules.git
4 1  [submodule "cmake"]
5 2     path = cmake
6 3     url = git://github.com/jrl-umi3218/jrl-cmakemodules.git
```

Figure 1-14 Pull request to fix the issue

## 6.1.4 Different standard between LAAS-CNRS and ROS

### Naming Standards

The naming style is different between the ROS build farm and the packages from the laboratory. When there is a package with more than one word it is separated by an underscore ( `_` ) in the ROS build farm and with hyphens ( `-` ) in the laboratory. For the first package, they requested me to change `dynamic-graph` into `dynamic_graph`. Most of the packages from the build farm are with underscores and the ROS build farm maintainer argued that it should be underscored to not break the build farm. However, some of the packages on the build farm are with hyphens like the `hpp-fcl` package that is also developed by the LAAS-CNRS.

If we have to change every name into packages from the laboratory it would take time to do the changes on every package name developed for 10 years and it would be error-prone.

After discussing with the ROS developer he agreed on the changes but was asking help on other ROS deputies. It took 20 days for a ROS deputy to give his help. During this time I advanced on other packages to fix the warnings and the dependencies so that they will be ready to launch when `dynamic-graph` will be released. He explained that the naming rule of the underscore is mandatory for ROS packages so that the ROS tool can handle it. However, `dynamic-graph` is not an official ROS package; it is a third-party one. Thanks to this difference the package is not using any ROS messages and therefore should not cause any issues in the future.

### Plugin management

The Stack of Tasks is using a plugin system that is different from the ROS build farm. The plugins are copied inside `/opt/ros/melodic/lib/`

However, we do not have access to write in this file inside the build farm. The most long term solution would be to change how the SoT behaves with plugins to match ros plugin behaviour. It would have been too much time consuming and error-prone therefore we choose another option. This option is to write the plugins elsewhere only on the buildfarm by using patches. Patches are

modifications that apply only on upstream code that are upload on the build-farm

### 6.1.5 Time delay

For each release step, it took more or less time to complete.

For the release of the packages for robot pkg, it takes approximately 1 to 2 days to complete because I can contact Guillhem Saurel directly. He is the engineer in charge of checking and releasing the packages. A release on robotpkg is mandatory before doing a release on the ROS build farm. The release of dynamic-graph-python was a little longer since other issues needed to be fixed in the same release.

As seen in the previous section it took 23 days to put dynamic-graph on the build farm because of standard difference. The release of a new fixed version of dynamic-graph and for other ROS version took between 1 day to 15 days.

To summarize the time of each step for each package for each version from the date I published the changes:

Name of package	ROS version	Release from the laboratory	Release from the build farm
Dynamic-graph	melodic	Less than 1 day	23 days
	melodic		4 days
Dynamic-graph(fixed)	noetic	15 days	1 day
	foxy		1 day
	eloquent		9 days
Dynamic-graph-python	melodic	4 days	Less than 1 day
	noetic		1 day
Dynamic-graph-tutorial	melodic	11 minutes	1.5 days
	noetic		1 day
Eiquadprog	melodic	6 days	2 days
	noetic		

TSID	foxy	4 days	1 hour 30 minutes
	eloquent		
	melodic		
	noetic		
sot-core	melodic	17 days	1 day
	noetic		

The delays are variables and depend on many things. It can be caused by people that are on holidays and can not see the release request or if the solution needs to have some changes, it needs some time to discuss it to be better.

The first packages are now on the build farm so the time delay will be much shorter for other packages. There will be no more discussion about the package naming standards and it will be the end of vacation for most of the workers.

### **Build farm errors and delays**

For dynamic-graph-tutorial, my package was put on hold because there were two faulty packages released by other developers of the ROS community that need to be fixed before syncing the build farm. When a case like this appears a discussion is open on the ROS discourse to keep the developers informed:

<https://discourse.ros.org/t/preparing-for-melodic-sync-2020-08-19/15988>

### **Double release of a package in robotpkg**

The process to release a package is optimized to incrementally add modification and test however if you want to do a minor release it is not made for this. To unlock my work Guilhem SAUREL tried to do the release with the package.xml file in the upstream repository however due to the complexity; this release broke the Stack of Tasks. I needed to wait for the major release to have the package ready.

## **6.2 Technical discussion**

It was hard to anticipate the errors in the original package since it should have been error-free because it was on the build farm before.

I was not expecting the time delays on the releases. Now I see that the dependencies are complex and need a global comprehension to work with since a mistake can break everything.

## 7 Conclusion and future work

This internship helped me to understand the research field and its objectives in term of development and way of working. I understood what was behind the scene of ROS build farm and all the work that is needed to release packages so that it is simpler to download for end-users.

It will also help the researchers at the laboratory because the first packages are on the ROS build farm and it will take little time for them to continue to maintain those packages. RIO project can now use some of the packages of the SoT on ROS and when all the packages will be released it will be possible to implement the TIAGO robot on the training platform.

For future work, I need to validate my tutorials with the researchers and finish to implement the last packages. When every package will be on the build farm I will be able to test them on the real TIAGO robot.



## REFERENCES

- [1] A. N. Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, "ROS: an open-source Robot Operating System."
- [2] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, "A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks," *2009 Int. Conf. Adv. Robot. ICAR 2009*, 2009.
- [3] M. Naveau, M. Kudruss, O. Stasse, C. Kirches, K. Mombaur, and P. Souères, "A Reactive Walking Pattern Generator Based on Nonlinear Model Predictive Control," *IEEE Robot. Autom. Lett.*, vol. 2, no. 1, pp. 10–17, 2017, doi: 10.1109/LRA.2016.2518739.
- [4] O. Stasse *et al.*, "TALOS : A new humanoid research platform targeted for industrial applications," 2018.
- [5] D. E. Domenichelli, S. Traversaro, L. Muratore, A. Rocchi, F. Nori, and L. Natale, "A Build System for Software Development in Robotic Academic Collaborative Environments," *Int. J. Semant. Comput.*, vol. 13, no. 2, pp. 185–205, 2019, doi: 10.1142/S1793351X19400087.
- [6] H. Bruyninckx, "Open robot control software: the OROCOS project," *IEEE Int. Conf. Robot. Automotion*, 2001.
- [7] J. A. Forbes, J. D. Stone, S. Parthasarathy, M. J. Toutonghi, and M. V. Sliger, "Software package management," vol. 2, no. 12, 2002, [Online]. Available: <https://patents.google.com/patent/US6381742B2/en>.
- [8] S. Clara and S. Clara, "( 12 ) United States Patent," vol. 1, no. 12, 2004.
- [9] M. A. Us, F. Schwichtenberg, M. A. Us, and J. Yasskin, "(12) Patent Application Publication (10) Pub. No.: US 2007/0101197 A1," vol. 1, no. 19, 2007.
- [10] A. N. A. Carolina, R. D. C. Gross, and A. P. C. Seca, "Patent Application Publication," *npm, Inc.; Isaac Z. Schlueter*, pp. 1–5, 2015.
- [11] J. Cappos and J. Samuel, "Package management security," *Univ. Arizona ...*, pp. 1–20, 2008, [Online]. Available: <http://www.cs.arizona.edu/people/jsamuel/papers/TR08-02.pdf>.
- [12] E. Y. Jean-Bernard Hayet, Claudia Esteves, Gustavo Arechavaleta-Servin, Olivier Stasse, "HUMANOID LOCOMOTION PLANNING FOR VISUALLY-GUIDED TASKS," 2014.

- [13] A. Nicolin, J. Mirabel, S. Boria, O. Stasse, and F. Lamiroux, "Agimus: A new framework for mapping manipulation motion plans to sequences of hierarchical task-based controllers," *Proc. 2020 IEEE/SICE Int. Symp. Syst. Integr. SII 2020*, pp. 1022–1027, 2020, doi: 10.1109/SII46433.2020.9026288.
- [14] A. Del Prete, N. Mansard, O. E. Ramos, O. Stasse, and F. Nori, "Implementing Torque Control with High-Ratio Gear Boxes and Without Joint-Torque Sensors," *Int. J. Humanoid Robot.*, vol. 13, no. 1, 2016, doi: 10.1142/S0219843615500449.
- [15] I. Ramirez-alpizar *et al.*, "Motion Generation for Pulling a Fire Hose by a Humanoid Robot," *ICHR*, 2016.
- [16] M. Karklinsky *et al.*, "Robust human-inspired power law trajectories for humanoid HRP-2 robot," *BioRob*, 2019.